RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing 日本語訳

原文URL : https://datatracker.ietf.org/doc/html/rfc7230

タイトル : **RFC 7230 - ハイパーテキスト転送プロトコル（HTTP / 1.1） ：メッセージの構文とルーティング**

翻訳編集 : 自動生成, ST: Proposed Standard, WG: tls

このRFCは廃止されました。修正版は RFC 9110, RFC 9112 です。


```
Internet Engineering Task Force (IETF)                  R. Fielding, Ed.
Request for Comments: 7230                                         Adobe
Obsoletes: 2145, 2616                                   J. Reschke, Ed.
Updates: 2817, 2818                                          greenbytes
Category: Standards Track                                     June 2014
ISSN: 2070-1721
```


Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing

ハイパーテキスト転送プロトコル（HTTP / 1.1） ：メッセージの構文とルーティング

Abstract

The Hypertext Transfer Protocol (HTTP) is a stateless application-level protocol for distributed, collaborative, hypertext information systems. This document provides an overview of HTTP architecture and its associated terminology, defines the "http" and "https" Uniform Resource Identifier (URI) schemes, defines the HTTP/1.1 message syntax and parsing requirements, and describes related security concerns for implementations.

概要

ハイパーテキスト転送プロトコル（HTTP）は、分散型の協調型ハイパーテキスト情報システム用のステートレスアプリケーションレベルプロトコルです。このドキュメントでは、HTTPアーキテクチャとそれに関連する用語の概要を説明し、「http」と「https」のURI（Uniform Resource Identifier）スキームを定義し、HTTP / 1.1メッセージの構文と解析要件を定義し、実装に関連するセキュリティ上の懸念について説明します。

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7230.

本文書の状態

これはInternet Standards Trackドキュメントです。

このドキュメントは、IETF（Internet Engineering Task Force）の製品です。これは、IETFコミュニティのコンセンサスを表しています。公開レビューを受け、インターネットエンジニアリンググステアリンググループ（IESG）による公開が承認されました。インターネット標準の詳細については、RFC 5741のセクション2をご覧ください。

このドキュメントの現在のステータス、エラータ、およびフィードバックの提供方法に関する情報は、http：//www.rfc-editor.org/info/rfc7230で入手できます。

Table of Contents

目次

1. Introduction

The Hypertext Transfer Protocol (HTTP) is a stateless application-level request/response protocol that uses extensible semantics and self-descriptive message payloads for flexible interaction with network-based hypertext information systems. This document is the first in a series of documents that collectively form the HTTP/1.1 specification:

1. "Message Syntax and Routing" (this document)

2. "Semantics and Content" [RFC7231]

3. "Conditional Requests" [RFC7232]

4. "Range Requests" [RFC7233]

5. "Caching" [RFC7234]

6. "Authentication" [RFC7235]

This HTTP/1.1 specification obsoletes RFC 2616 and RFC 2145 (on HTTP versioning). This specification also updates the use of CONNECT to establish a tunnel, previously defined in RFC 2817, and defines the "https" URI scheme that was described informally in RFC 2818.

HTTP is a generic interface protocol for information systems. It is designed to hide the details of how a service is implemented by presenting a uniform interface to clients that is independent of the types of resources provided. Likewise, servers do not need to be aware of each client's purpose: an HTTP request can be considered in isolation rather than being associated with a specific type of client or a predetermined sequence of application steps. The result is a protocol that can be used effectively in many different contexts and for which implementations can evolve independently over time.

HTTP is also designed for use as an intermediation protocol for

1. はじめに

ハイパーテキスト転送プロトコル（HTTP）はステートレスなアプリケーションレベルの要求/応答プロトコルであり、拡張可能なセマンティクスと自己記述的なメッセージペイロードを使用して、ネットワークベースのハイパーテキスト情報システムと柔軟に対話します。このドキュメントは、HTTP / 1.1仕様をまとめて形成する一連のドキュメントの最初のものです。

1. 「メッセージの構文とルーティング」（このドキュメント）

2. 「セマンティクスとコンテンツ」[RFC7231]

3. 「条件付きリクエスト」[RFC7232]

4. 「範囲リクエスト」[RFC7233]

5. 「キャッシング」[RFC7234]

6. 「認証」[RFC7235]

このHTTP / 1.1仕様は、RFC 2616およびRFC 2145（HTTPバージョニングについて）を廃止します。この仕様はまた、以前にRFC 2817で定義されたトンネルを確立するためのCONNECTの使用を更新し、RFC 2818で非公式に説明された「https」URIスキームを定義します。

HTTPは、情報システムの汎用インターフェースプロトコルです。提供されるリソースのタイプに依存しない均一なインターフェースをクライアントに提示することにより、サービスの実装方法の詳細を隠すように設計されています。同様に、サーバーは各クライアントの目的を認識する必要はありません。HTTP要求は、特定のタイプのクライアントや事前に定義された一連のアプリケーションステップに関連付けられるのではなく、単独で考慮されます。その結果、多くの異なるコンテキストで効果的に使用でき、実装が時間の経過とともに独立して進化できるプロトコルが得られます。

HTTPは、非HTTP情報システムとの間の通信を変換するための

translating communication to and from non-HTTP information systems. HTTP proxies and gateways can provide access to alternative information services by translating their diverse protocols into a hypertext format that can be viewed and manipulated by clients in the same way as HTTP services.

仲介プロトコルとして使用するためにも設計されています。HTTPプロキシとゲートウェイは、HTTPサービスと同じ方法でクライアントが表示および操作できるハイパーテキスト形式に多様なプロトコルを変換することにより、代替情報サービスへのアクセスを提供できます。

One consequence of this flexibility is that the protocol cannot be defined in terms of what occurs behind the interface. Instead, we are limited to defining the syntax of communication, the intent of received communication, and the expected behavior of recipients. If the communication is considered in isolation, then successful actions ought to be reflected in corresponding changes to the observable interface provided by servers. However, since multiple clients might act in parallel and perhaps at cross-purposes, we cannot require that such changes be observable beyond the scope of a single response.

この柔軟性の結果として、インターフェイスの背後で発生することに関してプロトコルを定義できなくなります。代わりに、通信の構文、受信した通信の意図、および受信者の予想される動作の定義に限定されます。通信が単独で考慮される場合、成功したアクションは、サーバーによって提供される監視可能なインターフェースへの対応する変更に反映されるべきです。ただし、複数のクライアントが並行して、また多目的に動作する可能性があるため、そのような変更が単一の応答の範囲を超えて監視可能であることを要求することはできません。

This document describes the architectural elements that are used or referred to in HTTP, defines the "http" and "https" URI schemes, describes overall network operation and connection management, and defines HTTP message framing and forwarding requirements. Our goal is to define all of the mechanisms necessary for HTTP message handling that are independent of message semantics, thereby defining the complete set of requirements for message parsers and message-forwarding intermediaries.

このドキュメントでは、HTTPで使用または参照されるアーキテクチャ要素について説明し、「http」と「https」のURIスキームを定義し、ネットワーク操作と接続管理全体を説明し、HTTPメッセージのフレーミングと転送の要件を定義します。私たちの目標は、メッセージのセマンティクスに依存しないHTTPメッセージ処理に必要なすべてのメカニズムを定義することです。これにより、メッセージパーサーとメッセージ転送中間体の要件の完全なセットを定義します。

1.1. Requirements Notation

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 [RFC2119].

1.1. 要件表記

このドキュメントのキーワード「MUST」、「MUST NOT」、「REQUIRED」、「SHALL」、「SHALL NOT」、「SHOULD」、「SHOULD NOT」、「RECOMMENDED」、「MAY」、および「OPTIONAL」は、[RFC2119]で説明されているように解釈されます。

Conformance criteria and considerations regarding error handling are defined in Section 2.5.

エラー処理に関する適合基準と考慮事項は、セクション2.5で定義されています。

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with a list extension, defined in Section 7, that allows for compact definition of comma-separated lists using a '#' operator (similar to how the '*' operator indicates repetition). Appendix B shows the collected grammar with all list operators expanded to standard ABNF notation.

1.2. 構文表記

この仕様では、セクション7で定義されているリスト拡張子を持つ[RFC5234]の拡張バッカスナウア記法（ABNF）表記を使用し、'#'演算子を使用してコンマ区切りのリストをコンパクトに定義できます（'*'演算子は繰り返しを示します）。付録Bは、すべてのリスト演算子が標準のABNF表記に拡張された、収集された文法を示しています。

The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), HTAB (horizontal tab), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible [USASCII] character).

[RFC5234]、付録B.1で定義されているように、次のコアルールが参照として含まれています。ALPHA（文字）、CR（キャリッジリターン）、CRLF（CR LF）、CTL（コントロール）、DIGIT（10進数の0-9）、DQUOTE（二重引用符）、HEXDIG（16進数の0-9 / AF / af）、HTAB（水平タブ）、LF（ラインフィード）、OCTET（データの任意の8ビットシーケンス）、SP（スペース）、およびVCHAR（表示される[USASCII]文字）。

As a convention, ABNF rule names prefixed with "obs-" denote "obsolete" grammar rules that appear for historical reasons.

慣例として、「obs-」で始まるABNFルール名は、歴史的な理由で表示される「廃止された」文法ルールを示します。

2. Architecture

HTTP was created for the World Wide Web (WWW) architecture and has evolved over time to support the scalability needs of a worldwide hypertext system. Much of that architecture is reflected in the terminology and syntax productions used to define HTTP.

2. 建築

HTTPは、World Wide Web（WWW）アーキテクチャ用に作成され、世界的なハイパーテキストシステムのスケーラビリティのニーズをサポートするために時間とともに進化してきました。そのアーキテクチャの多くは、HTTPの定義に使用される用語と構文の生成に反映されています。

2.1. Client/Server Messaging

HTTP is a stateless request/response protocol that operates by exchanging messages (Section 3) across a reliable transport- or session-layer "connection" (Section 6). An HTTP "client" is a program that establishes a connection to a server for the purpose of sending one or more HTTP requests. An HTTP "server" is a program that accepts connections in order to service HTTP requests by sending HTTP responses.

The terms "client" and "server" refer only to the roles that these programs perform for a particular connection. The same program might act as a client on some connections and a server on others. The term "user agent" refers to any of the various client programs that initiate a request, including (but not limited to) browsers, spiders (web-based robots), command-line tools, custom applications, and mobile apps. The term "origin server" refers to the program that can originate authoritative responses for a given target resource. The terms "sender" and "recipient" refer to any implementation that sends or receives a given message, respectively.

HTTP relies upon the Uniform Resource Identifier (URI) standard [RFC3986] to indicate the target resource (Section 5.1) and relationships between resources. Messages are passed in a format similar to that used by Internet mail [RFC5322] and the Multipurpose Internet Mail Extensions (MIME) [RFC2045] (see Appendix A of [RFC7231] for the differences between HTTP and MIME messages).

Most HTTP communication consists of a retrieval request (GET) for a representation of some resource identified by a URI. In the simplest case, this might be accomplished via a single bidirectional connection (===) between the user agent (UA) and the origin server (O).

```
          request   >
     UA ======================================= O
                                <   response
```

A client sends an HTTP request to a server in the form of a request message, beginning with a request-line that includes a method, URI, and protocol version (Section 3.1.1), followed by header fields containing request modifiers, client information, and representation metadata (Section 3.2), an empty line to indicate the end of the header section, and finally a message body containing the payload body (if any, Section 3.3).

A server responds to a client's request by sending one or more HTTP response messages, each beginning with a status line that includes the protocol version, a success or error code, and textual reason phrase (Section 3.1.2), possibly followed by header fields containing server information, resource metadata, and representation metadata (Section 3.2), an empty line to indicate the end of the header section, and finally a message body containing the payload body (if any, Section 3.3).

A connection might be used for multiple request/response exchanges, as defined in Section 6.3.

The following example illustrates a typical message exchange for a GET request (Section 4.3.1 of [RFC7231]) on the URI "http://www.example.com/hello.txt":

2.1. クライアント/サーバーメッセージング

HTTPは、信頼できるトランスポート層またはセッション層の「接続」（セクション6）全体でメッセージを交換することによって動作するステートレスの要求/応答プロトコル（セクション3）です。 HTTP「クライアント」は、1つ以上のHTTP要求を送信する目的でサーバーへの接続を確立するプログラムです。 HTTP「サーバー」は、HTTP応答を送信してHTTP要求を処理するために接続を受け入れるプログラムです。

「クライアント」および「サーバー」という用語は、これらのプログラムが特定の接続に対して実行する役割のみを指します。同じプログラムが一部の接続ではクライアントとして機能し、別の接続ではサーバーとして機能する場合があります。 「ユーザーエージェント」という用語は、ブラウザー、スパイダー（Webベースのロボット）、コマンドラインツール、カスタムアプリケーション、モバイルアプリ（これらに限定されない）を含む、リクエストを開始するさまざまなクライアントプログラムのいずれかを指します。 「発信元サーバー」という用語は、特定のターゲットリソースに対して信頼できる応答を発信できるプログラムを指します。 「送信者」および「受信者」という用語は、それぞれ特定のメッセージを送信または受信する実装を指します。

HTTPは、Uniform Resource Identifier（URI）標準[RFC3986]に依存して、ターゲットリソース（セクション5.1）とリソース間の関係を示します。メッセージは、インターネットメール[RFC5322]および多目的インターネットメール拡張（MIME）[RFC2045]で使用されるものと同様の形式で渡されます（HTTPメッセージとMIMEメッセージの違いについては、[RFC7231]の付録Aを参照してください）。

ほとんどのHTTP通信は、URIで識別されるリソースを表すための取得要求（GET）で構成されています。最も単純なケースでは、これはユーザーエージェント（UA）とオリジンサーバー（O）の間の単一の双方向接続（===）を介して行われる可能性があります。

クライアントは、メソッド、URI、およびプロトコルバージョン（セクション3.1.1）を含むリクエスト行で始まり、リクエスト修飾子を含むヘッダーフィールド、クライアント情報が続くリクエストメッセージの形式で、サーバーにHTTPリクエストを送信します。 、および表現メタデータ（セクション3.2）、ヘッダーセクションの終わりを示す空の行、最後にペイロードボディを含むメッセージボディ（セクション3.3）。

サーバーは、1つ以上のHTTP応答メッセージを送信することにより、クライアントの要求に応答します。各メッセージは、プロトコルバージョン、成功またはエラーコード、およびテキストの理由句（セクション3.1.2）を含むステータス行で始まり、ヘッダーフィールドが続く場合があります。サーバー情報、リソースメタデータ、表現メタデータ（セクション3.2）、ヘッダーセクションの終わりを示す空の行、ペイロード本文を含むメッセージ本文（セクション3.3）が含まれます。

セクション6.3で定義されているように、接続は複数の要求/応答交換に使用される場合があります。

次の例は、URI「http://www.example.com/hello.txt」でのGETリクエスト（[RFC7231]のセクション4.3.1）の一般的なメッセージ交換を示しています。

Client request:

クライアントのリクエスト：

```
GET /hello.txt HTTP/1.1
User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3
Host: www.example.com
Accept-Language: en, mi
```

Server response:

サーバーの応答：

```
HTTP/1.1 200 OK
Date: Mon, 27 Jul 2009 12:28:53 GMT
Server: Apache
Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT
ETag: "34aa387-d-1568eb00"
Accept-Ranges: bytes
Content-Length: 51
Vary: Accept-Encoding
Content-Type: text/plain
```

Hello World! My payload includes a trailing CRLF.

"こんにちは世界"私のペイロードには末尾のCRLFが含まれています。

2.2. Implementation Diversity

2.2. 実装の多様性

When considering the design of HTTP, it is easy to fall into a trap of thinking that all user agents are general-purpose browsers and all origin servers are large public websites. That is not the case in practice. Common HTTP user agents include household appliances, stereos, scales, firmware update scripts, command-line programs, mobile apps, and communication devices in a multitude of shapes and sizes. Likewise, common HTTP origin servers include home automation units, configurable networking components, office machines, autonomous robots, news feeds, traffic cameras, ad selectors, and video-delivery platforms.

HTTPの設計を検討するとき、すべてのユーザーエージェントは汎用ブラウザであり、すべてのオリジンサーバーは大規模なパブリックWebサイトであるという考えの罠に陥るのは簡単です。実際にはそうではありません。一般的なHTTPユーザーエージェントには、さまざまな形やサイズの家電、ステレオ、体重計、ファームウェア更新スクリプト、コマンドラインプログラム、モバイルアプリ、通信デバイスなどがあります。同様に、一般的なHTTPオリジンサーバーには、ホームオートメーションユニット、設定可能なネットワークコンポーネント、オフィスマシン、自律ロボット、ニュースフィード、交通カメラ、広告セレクター、ビデオ配信プラットフォームが含まれます。

The term "user agent" does not imply that there is a human user directly interacting with the software agent at the time of a request. In many cases, a user agent is installed or configured to run in the background and save its results for later inspection (or save only a subset of those results that might be interesting or erroneous). Spiders, for example, are typically given a start URI and configured to follow certain behavior while crawling the Web as a hypertext graph.

「ユーザーエージェント」という用語は、要求時にソフトウェアエージェントと直接対話する人間のユーザーがいることを意味するものではありません。多くの場合、ユーザーエージェントはバックグラウンドで実行されるようにインストールまたは構成され、後で検査するために結果を保存します（または、興味深いまたは誤っている可能性がある結果のサブセットのみを保存します）。たとえば、スパイダーには通常、開始URIが与えられ、Webをハイパーテキストグラフとしてクロールするときに特定の動作を追跡するように構成されます。

The implementation diversity of HTTP means that not all user agents can make interactive suggestions to their user or provide adequate warning for security or privacy concerns. In the few cases where this specification requires reporting of errors to the user, it is acceptable for such reporting to only be observable in an error console or log file. Likewise, requirements that an automated action be confirmed by the user before proceeding might be met via advance configuration choices, run-time options, or simple avoidance of the unsafe action; confirmation does not imply any specific user interface or interruption of normal processing if the user has already made that choice.

HTTPの実装の多様性は、すべてのユーザーエージェントがユーザーにインタラクティブな提案をしたり、セキュリティやプライバシーの懸念に対して適切な警告を提供したりできるわけではないことを意味します。この仕様がユーザーへのエラーの報告を必要とするいくつかのケースでは、そのような報告がエラーコンソールまたはログファイルでのみ観察可能であることは許容されます。同様に、続行する前にユーザーが自動化されたアクションを確認するという要件は、事前構成の選択、ランタイムオプション、または危険なアクションの単純な回避によって満たされる場合があります。ユーザーが既にその選択を行っている場合、確認は特定のユーザーインターフェイスや通常の処理の中断を意味するものではありません。

2.3. Intermediaries

2.3. 仲介人

HTTP enables the use of intermediaries to satisfy requests through a chain of connections. There are three common forms of HTTP intermediary: proxy, gateway, and tunnel. In some cases, a single intermediary might act as an origin server, proxy, gateway, or tunnel, switching behavior based on the nature of each request.

HTTPを使用すると、仲介者を使用して、一連の接続を通じて要求を満たすことができます。 HTTP仲介には、プロキシ、ゲートウェイ、トンネルの3つの一般的な形式があります。場合によっては、単一の仲介者がオリジンサーバー、プロキシ、ゲートウェイ、またはトンネルとして機能し、各リクエストの性質に基づいて動作を切り替えます。

```
      >             >             >             >
UA ========== A ========== B ========== C ========== O
           <             <             <             <
```

The figure above shows three intermediaries (A, B, and C) between the user agent and origin server. A request or response message that travels the whole chain will pass through four separate connections. Some HTTP communication options might apply only to the connection with the nearest, non-tunnel neighbor, only to the endpoints of the chain, or to all connections along the chain. Although the diagram is linear, each participant might be engaged in multiple, simultaneous communications. For example, B might be receiving requests from many clients other than A, and/or forwarding requests to servers other than C, at the same time that it is handling A's request. Likewise, later requests might be sent through a different path of connections, often based on dynamic configuration for load balancing.

上の図は、ユーザーエージェントと配信元サーバーの間の3つの仲介者（A、B、C）を示しています。チェーン全体を移動する要求または応答メッセージは、4つの個別の接続を通過します。一部のHTTP通信オプションは、最も近い非トンネルネイバーとの接続にのみ、チェーンのエンドポイントにのみ、またはチェーンに沿ったすべての接続に適用される場合があります。図は直線的ですが、各参加者は複数の同時通信に従事している場合があります。たとえば、BはA以外の多くのクライアントからの要求を受信したり、C以外のサーバーに要求を転送したりすると同時に、Aの要求を処理している可能性があります。同様に、後の要求は、多くの場合、ロードバランシングの動的構成に基づいて、別の接続パスを介して送信される可能性があります。

The terms "upstream" and "downstream" are used to describe directional requirements in relation to the message flow: all messages flow from upstream to downstream. The terms "inbound" and "outbound" are used to describe directional requirements in relation to the request route: "inbound" means toward the origin server and "outbound" means toward the user agent.

「アップストリーム」および「ダウンストリーム」という用語は、メッセージフローに関連する方向の要件を説明するために使用されます。すべてのメッセージはアップストリームからダウンストリームに流れます。 「インバウンド」および「アウトバウンド」という用語は、リクエストルートに関連する方向の要件を説明するために使用されます。「インバウンド」はオリジンサーバーに向かうことを意味し、「アウトバウンド」はユーザーエージェントに向かうことを意味します。

A "proxy" is a message-forwarding agent that is selected by the client, usually via local configuration rules, to receive requests for some type(s) of absolute URI and attempt to satisfy those requests via translation through the HTTP interface. Some translations are minimal, such as for proxy requests for "http" URIs, whereas other requests might require translation to and from entirely different application-level protocols. Proxies are often used to group an organization's HTTP requests through a common intermediary for the sake of security, annotation services, or shared caching. Some proxies are designed to apply transformations to selected messages or payloads while they are being forwarded, as described in Section 5.7.2.

「プロキシ」は、クライアントによって選択されたメッセージ転送エージェントであり、通常はローカル構成ルールを介して、特定のタイプの絶対URIの要求を受信し、HTTPインターフェースを介した変換を介してそれらの要求を満たすことを試みます。 「http」URIのプロキシリクエストなど、一部の変換は最小限ですが、その他のリクエストでは、まったく異なるアプリケーションレベルのプロトコルとの間の変換が必要になる場合があります。プロキシは、セキュリティ、注釈サービス、または共有キャッシュのために、一般的な中間手段を介して組織のHTTPリクエストをグループ化するためによく使用されます。セクション5.7.2で説明されているように、一部のプロキシは、転送中に選択されたメッセージまたはペイロードに変換を適用するように設計されています。

A "gateway" (a.k.a. "reverse proxy") is an intermediary that acts as an origin server for the outbound connection but translates received requests and forwards them inbound to another server or servers. Gateways are often used to encapsulate legacy or untrusted information services, to improve server performance through "accelerator" caching, and to enable partitioning or load balancing of HTTP services across multiple machines.

「ゲートウェイ」（別名「リバースプロキシ」）は、アウトバウンド接続のオリジンサーバーとして機能しますが、受信したリクエストを変換し、インバウンドで別のサーバーに転送します。ゲートウェイは、レガシーまたは信頼できない情報サービスのカプセル化、「アクセラレータ」キャッシングによるサーバーパフォーマンスの向上、複数のマシン間でのHTTPサービスのパーティショニングまたはロードバランシングを可能にするためによく使用されます。

All HTTP requirements applicable to an origin server also apply to the outbound communication of a gateway. A gateway communicates with inbound servers using any protocol that it desires, including private extensions to HTTP that are outside the scope of this specification. However, an HTTP-to-HTTP gateway that wishes to interoperate with third-party HTTP servers ought to conform to user agent requirements on the gateway's inbound connection.

オリジンサーバーに適用されるすべてのHTTP要件は、ゲートウェイのアウトバウンド通信にも適用されます。ゲートウェイは、この仕様の範囲外であるHTTPへのプライベート拡張を含め、必要なプロトコルを使用してインバウンドサーバーと通信します。ただし、サードパーティのHTTPサーバーと相互運用するHTTP-to-HTTPゲートウェイは、ゲートウェイのインバウンド接続のユーザーエージェント要件に準拠する必要があります。

A "tunnel" acts as a blind relay between two connections without changing the messages. Once active, a tunnel is not considered a party to the HTTP communication, though the tunnel might have been initiated by an HTTP request. A tunnel ceases to exist when both ends of the relayed connection are

「トンネル」は、メッセージを変更せずに2つの接続間のブラインドリレーとして機能します。いったんアクティブになると、トンネルはHTTP通信の当事者とは見なされませんが、トンネルはHTTP要求によって開始された可能性があります。リレーされた接続の両端が閉じられると、トンネルは存在しなくなりま

closed. Tunnels are used to extend a virtual connection through an intermediary, such as when Transport Layer Security (TLS, [RFC5246]) is used to establish confidential communication through a shared firewall proxy.

す。トンネルは、トランスポート層セキュリティ（TLS、[RFC5246]）を使用して共有ファイアウォールプロキシを介して機密通信を確立する場合など、仲介者を介して仮想接続を拡張するために使用されます。

The above categories for intermediary only consider those acting as participants in the HTTP communication. There are also intermediaries that can act on lower layers of the network protocol stack, filtering or redirecting HTTP traffic without the knowledge or permission of message senders. Network intermediaries are indistinguishable (at a protocol level) from a man-in-the-middle attack, often introducing security flaws or interoperability problems due to mistakenly violating HTTP semantics.

上記の仲介者のカテゴリは、HTTP通信の参加者として機能するもののみを考慮しています。ネットワークプロトコルスタックの下位層で動作し、メッセージ送信者の知識や許可なしにHTTPトラフィックをフィルタリングまたはリダイレクトできる仲介者もいます。ネットワーク仲介者は、中間者攻撃と（プロトコルレベルで）区別がつかず、誤ってHTTPセマンティクスに違反することにより、セキュリティの欠陥や相互運用性の問題を引き起こすことがよくあります。

For example, an "interception proxy" [RFC3040] (also commonly known as a "transparent proxy" [RFC1919] or "captive portal") differs from an HTTP proxy because it is not selected by the client. Instead, an interception proxy filters or redirects outgoing TCP port 80 packets (and occasionally other common port traffic). Interception proxies are commonly found on public network access points, as a means of enforcing account subscription prior to allowing use of non-local Internet services, and within corporate firewalls to enforce network usage policies.

たとえば、「インターセプトプロキシ」[RFC3040]（「トランスペアレントプロキシ」[RFC1919]または「キャプティブポータル」とも呼ばれる）は、クライアントによって選択されないため、HTTPプロキシとは異なります。代わりに、代行受信プロキシは、発信TCPポート80パケット（およびその他の一般的なポートトラフィック）をフィルタリングまたはリダイレクトします。傍受プロキシは、非ローカルインターネットサービスの使用を許可する前にアカウントサブスクリプションを適用する手段として、また企業のファイアウォール内でネットワーク使用ポリシーを実施する手段として、パブリックネットワークアクセスポイントによく見られます。

HTTP is defined as a stateless protocol, meaning that each request message can be understood in isolation. Many implementations depend on HTTP's stateless design in order to reuse proxied connections or dynamically load balance requests across multiple servers. Hence, a server MUST NOT assume that two requests on the same connection are from the same user agent unless the connection is secured and specific to that agent. Some non-standard HTTP extensions (e.g., [RFC4559]) have been known to violate this requirement, resulting in security and interoperability problems.

HTTPはステートレスプロトコルとして定義されています。つまり、各要求メッセージを個別に理解できます。多くの実装は、プロキシ接続を再利用したり、複数のサーバー間でリクエストを動的に負荷分散したりするために、HTTPのステートレス設計に依存しています。したがって、サーバーは、接続が保護され、そのエージェントに固有でない限り、同じ接続での2つの要求が同じユーザーエージェントからのものであると想定してはなりません（MUST NOT）。一部の非標準のHTTP拡張（[RFC4559]など）は、この要件に違反しており、セキュリティと相互運用性の問題を引き起こすことがわかっています。

2.4. Caches

2.4. キャッシュ

A "cache" is a local store of previous response messages and the subsystem that controls its message storage, retrieval, and deletion. A cache stores cacheable responses in order to reduce the response time and network bandwidth consumption on future, equivalent requests. Any client or server MAY employ a cache, though a cache cannot be used by a server while it is acting as a tunnel.

「キャッシュ」とは、以前の応答メッセージのローカルストアと、メッセージの保存、取得、削除を制御するサブシステムです。キャッシュは、将来の同等の要求での応答時間とネットワーク帯域幅の消費を削減するために、キャッシュ可能な応答を格納します。トンネルとして機能しているサーバーはキャッシュを使用できませんが、クライアントまたはサーバーはキャッシュを使用できます。

The effect of a cache is that the request/response chain is shortened if one of the participants along the chain has a cached response applicable to that request. The following illustrates the resulting chain if B has a cached copy of an earlier response from O (via C) for a request that has not been cached by UA or A.

キャッシュの効果は、チェーン上の参加者の1人がそのリクエストに適用可能なキャッシュされたレスポンスを持っている場合、リクエスト/レスポンスチェーンが短縮されることです。以下は、UAまたはAによってキャッシュされていないリクエストに対して、BがOからの（Cを介した）以前の応答のキャッシュされたコピーを持っている場合の結果のチェーンを示しています。

```
            >             >
       UA =========== A =========== B - - - - - - C - - - - - - O
                  <             <
```

A response is "cacheable" if a cache is allowed to store a copy of the response message for use in answering subsequent requests. Even when a response is cacheable, there might be additional constraints placed by the client or by the origin

キャッシュが後続の要求への応答に使用する応答メッセージのコピーを保存できる場合、応答は「キャッシュ可能」です。応答がキャッシュ可能な場合でも、キャッシュされた応答を特定の要求に使用できる場合に、クライアントまたはオリジンサー

server on when that cached response can be used for a particular request. HTTP requirements for cache behavior and cacheable responses are defined in Section 2 of [RFC7234].

There is a wide variety of architectures and configurations of caches deployed across the World Wide Web and inside large organizations. These include national hierarchies of proxy caches to save transoceanic bandwidth, collaborative systems that broadcast or multicast cache entries, archives of pre-fetched cache entries for use in off-line or high-latency environments, and so on.

2.5. Conformance and Error Handling

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. Additional (social) requirements are placed on implementations, resource owners, and protocol element registrations when they apply beyond the scope of a single communication.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP.

Conformance includes both the syntax and semantics of protocol elements. A sender MUST NOT generate protocol elements that convey a meaning that is known by that sender to be false. A sender MUST NOT generate protocol elements that do not match the grammar defined by the corresponding ABNF rules. Within a given message, a sender MUST NOT generate protocol elements or syntax alternatives that are only allowed to be generated by participants in other roles (i.e., a role that the sender does not have for that message).

When a received protocol element is parsed, the recipient MUST be able to parse any value of reasonable length that is applicable to the recipient's role and that matches the grammar defined by the corresponding ABNF rules. Note, however, that some received protocol elements might not be parsed. For example, an intermediary forwarding a message might parse a header-field into generic field-name and field-value components, but then forward the header field without further parsing inside the field-value.

HTTP does not have specific length limitations for many of its protocol elements because the lengths that might be appropriate will vary widely, depending on the deployment context and purpose of the implementation. Hence, interoperability between senders and recipients depends on shared expectations regarding what is a reasonable length for each protocol element. Furthermore, what is commonly understood to be a reasonable length for some protocol elements has changed over the course of the past two decades of HTTP use and is expected to continue changing in the future.

At a minimum, a recipient MUST be able to parse and process protocol element lengths that are at least as long as the values

バーによって追加の制約が課される場合があります。キャッシュ動作とキャッシュ可能な応答のHTTP要件は、[RFC7234]のセクション2で定義されています。

World Wide Web全体および大規模組織の内部には、さまざまな種類のアーキテクチャとキャッシュ構成が配備されています。これらには、大洋横断的な帯域幅を節約するプロキシキャッシュの全国階層、キャッシュエントリをブロードキャストまたはマルチキャストするコラボレーションシステム、オフラインまたは高レイテンシ環境で使用するためにプリフェッチされたキャッシュエントリのアーカイブなどが含まれます。

2.5. 適合性とエラー処理

この仕様は、HTTP通信における参加者の役割に応じた適合基準を対象としています。したがって、HTTP要件は、要件によって制約されている動作に応じて、送信者、受信者、クライアント、サーバー、ユーザーエージェント、仲介者、配信元サーバー、プロキシ、ゲートウェイ、またはキャッシュに課されます。実装、リソース所有者、およびプロトコル要素の登録が単一の通信の範囲を超えて適用される場合、追加の（社会的）要件が課されます。

動詞「生成」は、「送信」の代わりに使用されます。この場合、要件は、プロトコル要素の作成と単に受信した要素を下流に転送することを区別します。

実装は、HTTPに関与するロールに関連するすべての要件に準拠している場合、準拠していると見なされます。

適合性には、プロトコル要素の構文とセマンティクスの両方が含まれます。送信者は、その送信者が偽であると認識している意味を伝えるプロトコル要素を生成してはいけません（MUST NOT）。送信者は、対応するABNFルールで定義された文法に一致しないプロトコル要素を生成してはならない（MUST NOT）。与えられたメッセージ内で、送信者は、他の役割（つまり、そのメッセージに対して送信者が持たない役割）の参加者のみが生成することを許可されるプロトコル要素または代替構文を生成してはなりません（MUST NOT）。

受信したプロトコル要素が解析されるとき、受信者は、受信者の役割に適用可能で、対応するABNFルールによって定義された文法に一致する妥当な長さの値を解析できなければなりません（MUST）。ただし、一部の受信プロトコル要素は解析されない場合があることに注意してください。たとえば、メッセージを転送する中間者は、ヘッダーフィールドを解析して一般的なフィールド名とフィールド値のコンポーネントに変換しますが、フィールド値の内部でさらに解析することなくヘッダーフィールドを転送します。

適切な長さは、デプロイメントのコンテキストと実装の目的に応じて大きく異なるため、HTTPのプロトコル要素の多くには特定の長さ制限はありません。したがって、送信者と受信者の間の相互運用性は、各プロトコル要素の妥当な長さに関する共通の期待に依存します。さらに、一部のプロトコル要素について妥当な長さであると一般的に理解されているものは、過去20年間のHTTPの使用の過程で変化し、将来も変化し続けることが予想されます。

少なくとも、受信者は、少なくとも他のメッセージの同じプロトコル要素に対して生成する値と同じ長さのプロトコル要素の

that it generates for those same protocol elements in other messages. For example, an origin server that publishes very long URI references to its own resources needs to be able to parse and process those same references when received as a request target.

長さを解析および処理できる必要があります。たとえば、独自のリソースへの非常に長いURI参照を公開するオリジンサーバーは、リクエストターゲットとして受信されたときに、同じ参照を解析および処理できる必要があります。

A recipient MUST interpret a received protocol element according to the semantics defined for it by this specification, including extensions to this specification, unless the recipient has determined (through experience or configuration) that the sender incorrectly implements what is implied by those semantics. For example, an origin server might disregard the contents of a received Accept-Encoding header field if inspection of the User-Agent header field indicates a specific implementation version that is known to fail on receipt of certain content codings.

受信者は、受信者が（経験または構成を通じて）送信者がそれらのセマンティクスが暗示するものを誤って実装していると判断しない限り、この仕様の拡張を含め、この仕様で定義されているセマンティクスに従って受信したプロトコル要素を解釈する必要があります。たとえば、User-Agentヘッダーフィールドの検査で特定のコンテンツコーディングの受信時に失敗することがわかっている特定の実装バージョンが示されている場合、オリジンサーバーは受信したAccept-Encodingヘッダーフィールドの内容を無視する可能性があります。

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

特に明記しない限り、受信者は、無効な構成から使用可能なプロトコル要素を回復しようと試みる場合があります。 HTTPは、セキュリティに直接影響がある場合を除いて、特定のエラー処理メカニズムを定義していません。プロトコルの異なるアプリケーションが異なるエラー処理戦略を必要とするためです。たとえば、Webブラウザーは、ロケーションヘッダーフィールドがABNFに従って解析されない応答から透過的に回復したい場合がありますが、システム制御クライアントは、あらゆる形式のエラー回復を危険であると見なす場合があります。

2.6. Protocol Versioning

2.6. プロトコルのバージョン管理

HTTP uses a "<major>.<minor>" numbering scheme to indicate versions of the protocol. This specification defines version "1.1". The protocol version as a whole indicates the sender's conformance with the set of requirements laid out in that version's corresponding specification of HTTP.

HTTPは "<major>。<minor>"番号付けスキームを使用して、プロトコルのバージョンを示します。この仕様はバージョン「1.1」を定義しています。全体としてのプロトコルバージョンは、そのバージョンの対応するHTTPの仕様で定められた一連の要件への送信者の適合を示します。

The version of an HTTP message is indicated by an HTTP-version field in the first line of the message. HTTP-version is case-sensitive.

HTTPメッセージのバージョンは、メッセージの最初の行のHTTP-versionフィールドで示されます。 HTTPバージョンでは大文字と小文字が区別されます。

```
    HTTP-version  = HTTP-name "/" DIGIT "." DIGIT
    HTTP-name     = %x48.54.54.50 ; "HTTP", case-sensitive
```

The HTTP version number consists of two decimal digits separated by a "." (period or decimal point). The first digit ("major version") indicates the HTTP messaging syntax, whereas the second digit ("minor version") indicates the highest minor version within that major version to which the sender is conformant and able to understand for future communication. The minor version advertises the sender's communication capabilities even when the sender is only using a backwards-compatible subset of the protocol, thereby letting the recipient know that more advanced features can be used in response (by servers) or in future requests (by clients).

HTTPバージョン番号は、「。」で区切られた2つの10進数で構成されます。 （ピリオドまたは小数点）。 1桁目（「メジャーバージョン」）はHTTPメッセージング構文を示し、2桁目（「マイナーバージョン」）は、送信者が準拠し、将来の通信のために理解できるメジャーバージョン内の最高のマイナーバージョンを示します。マイナーバージョンは、送信者がプロトコルの下位互換性のあるサブセットのみを使用している場合でも、送信者の通信機能をアドバタイズします。これにより、より高度な機能が応答（サーバー）または将来の要求（クライアント）で使用できることを受信者に通知します。 。

When an HTTP/1.1 message is sent to an HTTP/1.0 recipient [RFC1945] or a recipient whose version is unknown, the HTTP/1.1 message is constructed such that it can be interpreted as a valid HTTP/1.0 message if all of the newer features are ignored. This specification places recipient-version requirements on some new features so that a conformant sender will only use compatible features until it has determined, through configuration or the receipt of a message, that the recipient supports HTTP/1.1.

HTTP / 1.1メッセージがHTTP / 1.0受信者[RFC1945]またはバージョンが不明な受信者に送信される場合、HTTP / 1.1メッセージは、すべての新しいメッセージが有効なHTTP / 1.0メッセージとして解釈されるように構築されます。機能は無視されます。この仕様では、受信者バージョンの要件をいくつかの新機能に課しているため、構成またはメッセージの受信を通じて、受信者がHTTP / 1.1をサポートしていると判断するまで、適合送信者は互換機能のみを使用します。

The interpretation of a header field does not change between minor versions of the same major HTTP version, though the default behavior of a recipient in the absence of such a field can change. Unless specified otherwise, header fields defined in HTTP/1.1 are defined for all versions of HTTP/1.x. In particular, the Host and Connection header fields ought to be implemented by all HTTP/1.x implementations whether or not they advertise conformance with HTTP/1.1.

ヘッダーフィールドの解釈は、同じメジャーHTTPバージョンのマイナーバージョン間では変わりませんが、そのようなフィールドがない場合の受信者のデフォルトの動作は変わる可能性があります。特に明記されていない限り、HTTP / 1.1で定義されているヘッダーフィールドは、HTTP / 1.xのすべてのバージョンに対して定義されています。特に、HostおよびConnectionヘッダーフィールドは、HTTP / 1.1への準拠をアドバタイズするかどうかに関係なく、すべてのHTTP / 1.x実装で実装する必要があります。

New header fields can be introduced without changing the protocol version if their defined semantics allow them to be safely ignored by recipients that do not recognize them. Header field extensibility is discussed in Section 3.2.1.

定義されたセマンティクスにより、それらを認識しない受信者が安全に無視できる場合、プロトコルバージョンを変更せずに新しいヘッダーフィールドを導入できます。ヘッダーフィールドの拡張性については、セクション3.2.1で説明します。

Intermediaries that process HTTP messages (i.e., all intermediaries other than those acting as tunnels) MUST send their own HTTP-version in forwarded messages. In other words, they are not allowed to blindly forward the first line of an HTTP message without ensuring that the protocol version in that message matches a version to which that intermediary is conformant for both the receiving and sending of messages. Forwarding an HTTP message without rewriting the HTTP-version might result in communication errors when downstream recipients use the message sender's version to determine what features are safe to use for later communication with that sender.

HTTPメッセージを処理する仲介者（つまり、トンネルとして機能する仲介者以外のすべての仲介者）は、転送されたメッセージで独自のHTTPバージョンを送信する必要があります。つまり、HTTPメッセージの最初の行を盲目的に転送することはできません。そのメッセージのプロトコルバージョンが、メッセージの受信と送信の両方で仲介者が準拠しているバージョンと一致していることを確認する必要があります。 HTTPバージョンを書き換えずにHTTPメッセージを転送すると、ダウンストリーム受信者がメッセージ送信者のバージョンを使用して、その送信者とのその後の通信に安全に使用できる機能を決定するときに、通信エラーが発生する可能性があります。

A client SHOULD send a request version equal to the highest version to which the client is conformant and whose major version is no higher than the highest version supported by the server, if this is known. A client MUST NOT send a version to which it is not conformant.

クライアントは、クライアントが準拠している最高バージョンに等しいリクエストバージョンを送信する必要があります（SHOULD）。そのメジャーバージョンは、サーバーでサポートされている最高バージョン（既知の場合）以下です。クライアントは、準拠していないバージョンを送信してはなりません（MUST NOT）。

A client MAY send a lower request version if it is known that the server incorrectly implements the HTTP specification, but only after the client has attempted at least one normal request and determined from the response status code or header fields (e.g., Server) that the server improperly handles higher request versions.

サーバーがHTTP仕様を正しく実装していないことがわかっている場合、クライアントはより低い要求バージョンを送信できますが、クライアントが少なくとも1つの通常の要求を試み、応答ステータスコードまたはヘッダーフィールド（例：サーバー）からサーバーは、より高い要求バージョンを不適切に処理します。

A server SHOULD send a response version equal to the highest version to which the server is conformant that has a major version less than or equal to the one received in the request. A server MUST NOT send a version to which it is not conformant. A server can send a 505 (HTTP Version Not Supported) response if it wishes, for any reason, to refuse service of the client's major protocol version.

サーバーは、サーバーが準拠している最も高いバージョンと同じ応答バージョンを送信する必要があります（SHOULDは、要求で受信したメジャーバージョン以下のメジャーバージョンを持っています）。サーバーは、準拠していないバージョンを送信してはなりません（MUST NOT）。サーバーは、何らかの理由でクライアントのメジャープロトコルバージョンのサービスを拒否したい場合は、505（HTTPバージョンはサポートされていません）応答を送信できます。

A server MAY send an HTTP/1.0 response to a request if it is known or suspected that the client incorrectly implements the HTTP specification and is incapable of correctly processing later version responses, such as when a client fails to parse the version number correctly or when an intermediary is known to blindly forward the HTTP-version even when it doesn't conform to the given minor version of the protocol. Such protocol downgrades SHOULD NOT be performed unless triggered by specific client attributes, such as when one or more of the request header fields (e.g., User-Agent) uniquely match the values sent by a client known to be in error.

クライアントがHTTP仕様を誤って実装していて、クライアントがバージョン番号を正しく解析できない場合や、クライアントがバージョン番号を正しく解析できない場合など、後のバージョン応答を正しく処理できないことがわかっているか疑われる場合、サーバーは要求にHTTP / 1.0応答を送信できます仲介者は、指定されたマイナーバージョンのプロトコルに準拠していない場合でも、HTTPバージョンを盲目的に転送することが知られています。このようなプロトコルダウングレードは、1つ以上のリクエストヘッダーフィールド（User-Agentなど）が、エラーが発生していることがわかっているクライアントから送信された値と一意に一致する場合など、特定のクライアント属性によってトリガーされない限り、実行しないでください。

The intention of HTTP's versioning design is that the major

HTTPのバージョン管理設計の意図は、互換性のないメッセージ

number will only be incremented if an incompatible message syntax is introduced, and that the minor number will only be incremented when changes made to the protocol have the effect of adding to the message semantics or implying additional capabilities of the sender. However, the minor version was not incremented for the changes introduced between [RFC2068] and [RFC2616], and this revision has specifically avoided any such changes to the protocol.

構文が導入された場合にのみメジャー番号が増分され、プロトコルに加えられた変更がメッセージセマンティクスに追加または追加を示唆する効果を持つ場合にのみマイナー番号が増分されることです。送信者の機能。ただし、マイナーバージョンは、[RFC2068]と[RFC2616]の間に導入された変更については増分されず、この改訂では、プロトコルに対するそのような変更は特に回避されました。

When an HTTP message is received with a major version number that the recipient implements, but a higher minor version number than what the recipient implements, the recipient SHOULD process the message as if it were in the highest minor version within that major version to which the recipient is conformant. A recipient can assume that a message with a higher minor version, when sent to a recipient that has not yet indicated support for that higher version, is sufficiently backwards-compatible to be safely processed by any implementation of the same major version.

受信者が実装するメジャーバージョン番号を含むHTTPメッセージが受信されたが、受信者が実装するものよりも大きいマイナーバージョン番号である場合、受信者は、そのメジャーバージョン内の最も高いマイナーバージョンであるかのようにメッセージを処理する必要があります（SHOULD）。受信者は適合しています。受信者は、より高いマイナーバージョンのメッセージが、そのより高いバージョンのサポートをまだ示していない受信者に送信された場合、同じメジャーバージョンの実装で安全に処理できるように十分な下位互換性があると想定できます。

2.7. Uniform Resource Identifiers

2.7. Uniform Resource Identifiers

Uniform Resource Identifiers (URIs) [RFC3986] are used throughout HTTP as the means for identifying resources (Section 2 of [RFC7231]). URI references are used to target requests, indicate redirects, and define relationships.

Uniform Resource Identifiers（URIs）[RFC3986]は、リソースを識別する手段としてHTTP全体で使用されます（[RFC7231]のセクション2）。 URI参照は、リクエストのターゲット設定、リダイレクトの指定、関係の定義に使用されます。

The definitions of "URI-reference", "absolute-URI", "relative-part", "scheme", "authority", "port", "host", "path-abempty", "segment", "query", and "fragment" are adopted from the URI generic syntax. An "absolute-path" rule is defined for protocol elements that can contain a non-empty path component. (This rule differs slightly from the path-abempty rule of RFC 3986, which allows for an empty path to be used in references, and path-absolute rule, which does not allow paths that begin with "//".) A "partial-URI" rule is defined for protocol elements that can contain a relative URI but not a fragment component.

「URI-reference」、「absolute-URI」、「relative-part」、「scheme」、「authority」、「port」、「host」、「path-abempty」、「segment」、「query」の定義、および「フラグメント」は、URIの一般的な構文から採用されています。 「absolute-path」ルールは、空でないパスコンポーネントを含むことができるプロトコル要素に対して定義されます。 （この規則は、参照で空のパスを使用できるようにするRFC 3986のpath-abempty規則、および「//」で始まるパスを許可しないpath-absolute規則とは少し異なります。） 「URI」ルールは、相対URIを含むことができるがフラグメントコンポーネントは含まないプロトコル要素に対して定義されます。

```
    URI-reference = <URI-reference, see [RFC3986], Section 4.1>
    absolute-URI  = <absolute-URI, see [RFC3986], Section 4.3>
    relative-part = <relative-part, see [RFC3986], Section 4.2>
    scheme        = <scheme, see [RFC3986], Section 3.1>
    authority     = <authority, see [RFC3986], Section 3.2>
    uri-host      = <host, see [RFC3986], Section 3.2.2>
    port          = <port, see [RFC3986], Section 3.2.3>
    path-abempty  = <path-abempty, see [RFC3986], Section 3.3>
    segment       = <segment, see [RFC3986], Section 3.3>
    query         = <query, see [RFC3986], Section 3.4>
    fragment      = <fragment, see [RFC3986], Section 3.5>
```

```
    absolute-path = 1*( "/" segment )
    partial-URI   = relative-part [ "?" query ]
```

Each protocol element in HTTP that allows a URI reference will indicate in its ABNF production whether the element allows any form of reference (URI-reference), only a URI in absolute form (absolute-URI), only the path and optional query components, or some combination of the above. Unless otherwise indicated, URI references are parsed relative to the effective request URI (Section 5.5).

URI参照を許可するHTTPの各プロトコル要素は、その要素が任意の形式の参照（URI参照）、絶対形式のURIのみ（絶対URI）、パスとオプションのクエリコンポーネントのみを許可するかどうかをABNFプロダクションで示します。または上記のいくつかの組み合わせ。特に明記されていない限り、URI参照は有効なリクエストURIに基づいて解析されます（セクション5.5）。

2.7.1. http URI Scheme

2.7.1. http URIスキーム

The "http" URI scheme is hereby defined for the purpose of minting identifiers according to their association with the hierarchical namespace governed by a potential HTTP origin

「http」URIスキームは、特定のポートでTCP（[RFC0793]）接続をリッスンする潜在的なHTTPオリジンサーバーによって管理される階層的な名前空間との関連付けに従って識別子を作成す

server listening for TCP ([RFC0793]) connections on a given port.

る目的で定義されています。

```
    http-URI = "http:" "//" authority path-abempty [ "?" query ]
               [ "#" fragment ]
```

The origin server for an "http" URI is identified by the authority component, which includes a host identifier and optional TCP port ([RFC3986], Section 3.2.2). The hierarchical path component and optional query component serve as an identifier for a potential target resource within that origin server's name space. The optional fragment component allows for indirect identification of a secondary resource, independent of the URI scheme, as defined in Section 3.5 of [RFC3986].

「http」URIの配信元サーバーは、ホスト識別子とオプションのTCPポートを含む機関コンポーネントによって識別されます（[RFC3986]、セクション3.2.2）。階層パスコンポーネントとオプションのクエリコンポーネントは、そのオリジンサーバーの名前空間内の潜在的なターゲットリソースの識別子として機能します。オプションのフラグメントコンポーネントを使用すると、[RFC3986]のセクション3.5で定義されているように、URIスキームに関係なく、セカンダリリソースを間接的に識別できます。

A sender MUST NOT generate an "http" URI with an empty host identifier. A recipient that processes such a URI reference MUST reject it as invalid.

送信者は、ホスト識別子が空の "http" URIを生成してはならない（MUST NOT）。このようなURI参照を処理する受信者は、それを無効として拒否する必要があります。

If the host identifier is provided as an IP address, the origin server is the listener (if any) on the indicated TCP port at that IP address. If host is a registered name, the registered name is an indirect identifier for use with a name resolution service, such as DNS, to find an address for that origin server. If the port subcomponent is empty or not given, TCP port 80 (the reserved port for WWW services) is the default.

ホスト識別子がIPアドレスとして提供されている場合、オリジンサーバーは、そのIPアドレスで示されたTCPポートのリスナー（存在する場合）です。ホストが登録名である場合、登録名は、DNSなどの名前解決サービスでそのオリジンサーバーのアドレスを見つけるために使用する間接識別子です。ポートサブコンポーネントが空または指定されていない場合、TCPポート80（WWWサービス用に予約されているポート）がデフォルトです。

Note that the presence of a URI with a given authority component does not imply that there is always an HTTP server listening for connections on that host and port. Anyone can mint a URI. What the authority component determines is who has the right to respond authoritatively to requests that target the identified resource. The delegated nature of registered names and IP addresses creates a federated namespace, based on control over the indicated host and port, whether or not an HTTP server is present. See Section 9.1 for security considerations related to establishing authority.

特定の権限コンポーネントを持つURIの存在は、そのホストとポートで接続をリッスンしているHTTPサーバーが常に存在することを意味するわけではないことに注意してください。誰でもURIを作成できます。権限コンポーネントが決定するのは、識別されたリソースを対象とする要求に正式に応答する権利を持つ人です。登録された名前とIPアドレスの委任された性質により、HTTPサーバーが存在するかどうかに関係なく、指定されたホストとポートの制御に基づいて、フェデレーテッド名前空間が作成されます。権限の確立に関連するセキュリティの考慮事項については、セクション9.1を参照してください。

When an "http" URI is used within a context that calls for access to the indicated resource, a client MAY attempt access by resolving the host to an IP address, establishing a TCP connection to that address on the indicated port, and sending an HTTP request message (Section 3) containing the URI's identifying data (Section 5) to the server. If the server responds to that request with a non-interim HTTP response message, as described in Section 6 of [RFC7231], then that response is considered an authoritative answer to the client's request.

示されたリソースへのアクセスを要求するコンテキスト内で「http」URIが使用される場合、クライアントは、ホストをIPアドレスに解決し、示されたポートでそのアドレスへのTCP接続を確立し、HTTPを送信することにより、アクセスを試みることができます。サーバーへのURIの識別データ（セクション5）を含む要求メッセージ（セクション3）。[RFC7231]のセクション6で説明されているように、サーバーがその要求に非中間HTTP応答メッセージで応答する場合、その応答はクライアントの要求に対する信頼できる応答と見なされます。

Although HTTP is independent of the transport protocol, the "http" scheme is specific to TCP-based services because the name delegation process depends on TCP for establishing authority. An HTTP service based on some other underlying connection protocol would presumably be identified using a different URI scheme, just as the "https" scheme (below) is used for resources that require an end-to-end secured connection. Other protocols might also be used to provide access to "http" identified resources -- it is only the authoritative interface that is specific to TCP.

HTTPはトランスポートプロトコルから独立していますが、名前の委任プロセスは権限の確立をTCPに依存しているため、「http」スキームはTCPベースのサービスに固有です。エンドツーエンドのセキュリティで保護された接続を必要とするリソースに「https」スキーム（以下）が使用されるのと同じように、他の基になる接続プロトコルに基づくHTTPサービスは、おそらく別のURIスキームを使用して識別されます。 「http」で識別されたリソースへのアクセスを提供するために、他のプロトコルも使用される可能性があります。TCPに固有ののは信頼できるインターフェイスのみです。

The URI generic syntax for authority also includes a deprecated userinfo subcomponent ([RFC3986], Section 3.2.1) for including user authentication information in the URI. Some implementations make use of the userinfo component for

権限のURI汎用構文には、URIにユーザー認証情報を含めるための非推奨のuserinfoサブコンポーネント（[RFC3986]、セクション3.2.1）も含まれています。一部の実装では、コマンド呼び出しオプション、構成ファイル、ブックマークリスト内など、認

internal configuration of authentication information, such as within command invocation options, configuration files, or bookmark lists, even though such usage might expose a user identifier or password. A sender MUST NOT generate the userinfo subcomponent (and its "@" delimiter) when an "http" URI reference is generated within a message as a request target or header field value. Before making use of an "http" URI reference received from an untrusted source, a recipient SHOULD parse for userinfo and treat its presence as an error; it is likely being used to obscure the authority for the sake of phishing attacks.

証情報の内部構成にuserinfoコンポーネントを使用しますが、そのような使用法ではユーザー識別子またはパスワードが公開される可能性があります。メッセージ内に「http」URI参照が要求ターゲットまたはヘッダーフィールド値として生成される場合、送信者はuserinfoサブコンポーネント（およびその「@」区切り文字）を生成してはなりません（MUST NOT）。信頼できないソースから受信した「http」URI参照を利用する前に、受信者はuserinfoを解析して、その存在をエラーとして処理する必要があります（SHOULD）。フィッシング攻撃のために権限を隠すために使用されている可能性があります。

2.7.2. https URI Scheme

2.7.2. https URIスキーム

The "https" URI scheme is hereby defined for the purpose of minting identifiers according to their association with the hierarchical namespace governed by a potential HTTP origin server listening to a given TCP port for TLS-secured connections ([RFC5246]).

「https」URIスキームは、TLSで保護された接続（[RFC5246]）の特定のTCPポートをリッスンする潜在的なHTTPオリジンサーバーによって管理される階層的な名前空間との関連付けに従って識別子を作成するために定義されています。

All of the requirements listed above for the "http" scheme are also requirements for the "https" scheme, except that TCP port 443 is the default if the port subcomponent is empty or not given, and the user agent MUST ensure that its connection to the origin server is secured through the use of strong encryption, end-to-end, prior to sending the first HTTP request.

上記の「http」スキームのすべての要件は、「https」スキームの要件でもあります。ただし、ポートサブコンポーネントが空であるか指定されていない場合、TCPポート443がデフォルトであり、ユーザーエージェントは、元のサーバーは、最初のHTTP要求を送信する前に、エンドツーエンドで強力な暗号化を使用して保護されます。

```
    https-URI = "https:" "//" authority path-abempty [ "?" query ]
                [ "#" fragment ]
```

Note that the "https" URI scheme depends on both TLS and TCP for establishing authority. Resources made available via the "https" scheme have no shared identity with the "http" scheme even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are distinct namespaces and are considered to be distinct origin servers. However, an extension to HTTP that is defined to apply to entire host domains, such as the Cookie protocol [RFC6265], can allow information set by one service to impact communication with other services within a matching group of host domains.

「https」URIスキームは、権限を確立するためにTLSとTCPの両方に依存することに注意してください。 「https」スキームを介して利用可能にされたリソースは、リソース識別子が同じ権限（同じTCPポートをリッスンしている同じホスト）を示していても、「http」スキームとの共有IDを持ちません。それらは別個の名前空間であり、別個のオリジンサーバーと見なされます。ただし、Cookieプロトコル[RFC6265]など、ホストドメイン全体に適用するように定義されたHTTPの拡張機能により、1つのサービスによって設定された情報が、ホストドメインの一致するグループ内の他のサービスとの通信に影響を与える可能性があります。

The process for authoritative access to an "https" identified resource is defined in [RFC2818].

「https」で識別されたリソースへの権限のあるアクセスのプロセスは、[RFC2818]で定義されています。

2.7.3. http and https URI Normalization and Comparison

2.7.3. httpおよびhttps URIの正規化と比較

Since the "http" and "https" schemes conform to the URI generic syntax, such URIs are normalized and compared according to the algorithm defined in Section 6 of [RFC3986], using the defaults described above for each scheme.

「http」と「https」のスキームはURIの一般的な構文に準拠しているため、これらのURIは[RFC3986]のセクション6で定義されたアルゴリズムに従って正規化され、各スキームについて上記のデフォルトを使用して比較されます。

If the port is equal to the default port for a scheme, the normal form is to omit the port subcomponent. When not being used in absolute form as the request target of an OPTIONS request, an empty path component is equivalent to an absolute path of "/", so the normal form is to provide a path of "/" instead. The scheme and host are case-insensitive and normally provided in lowercase; all other components are compared in a case-sensitive manner. Characters other than those in the "reserved" set are equivalent to their percent-encoded octets: the normal form is to not encode them (see Sections 2.1 and 2.2 of [RFC3986]).

ポートがスキームのデフォルトポートと等しい場合、通常の形式はポートサブコンポーネントを省略します。 OPTIONSリクエストのリクエストターゲットとして絶対形式で使用されていない場合、空のパスコンポーネントは「/」の絶対パスに相当するため、通常の形式では「/」のパスを指定します。スキームとホストは大文字と小文字を区別せず、通常は小文字で提供されます。他のすべてのコンポーネントは、大文字と小文字を区別して比較されます。 「予約済み」セットの文字以外の文字は、パーセントでエンコードされたオクテットに相当します。通常の形式は、それらをエンコードしないことです（[RFC3986]のセクション2.1および2.2を参照）。

For example, the following three URIs are equivalent:

たとえば、次の3つのURIは同等です。

```
http://example.com:80/~smith/home.html
http://EXAMPLE.com/%7Esmith/home.html
http://EXAMPLE.com:/%7esmith/home.html
```

3. Message Format

All HTTP/1.1 messages consist of a start-line followed by a sequence of octets in a format similar to the Internet Message Format [RFC5322]: zero or more header fields (collectively referred to as the "headers" or the "header section"), an empty line indicating the end of the header section, and an optional message body.

HTTP-message = start-line *(header-field CRLF) CRLF [message-body]

The normal procedure for parsing an HTTP message is to read the start-line into a structure, read each header field into a hash table by field name until the empty line, and then use the parsed data to determine if a message body is expected. If a message body has been indicated, then it is read as a stream until an amount of octets equal to the message body length is read or the connection is closed.

A recipient MUST parse an HTTP message as a sequence of octets in an encoding that is a superset of US-ASCII [USASCII]. Parsing an HTTP message as a stream of Unicode characters, without regard for the specific encoding, creates security vulnerabilities due to the varying ways that string processing libraries handle invalid multibyte character sequences that contain the octet LF (%x0A). String-based parsers can only be safely used within protocol elements after the element has been extracted from the message, such as within a header field-value after message parsing has delineated the individual fields.

An HTTP message can be parsed as a stream for incremental processing or forwarding downstream. However, recipients cannot rely on incremental delivery of partial messages, since some implementations will buffer or delay message forwarding for the sake of network efficiency, security checks, or payload transformations.

A sender MUST NOT send whitespace between the start-line and the first header field. A recipient that receives whitespace between the start-line and the first header field MUST either reject the message as invalid or consume each whitespace-preceded line without further processing of it (i.e., ignore the entire line, along with any subsequent lines preceded by whitespace, until a properly formed header field is received or the header section is terminated).

The presence of such whitespace in a request might be an attempt to trick a server into ignoring that field or processing the line after it as a new request, either of which might result in a security vulnerability if other implementations within the request chain interpret the same message differently. Likewise, the presence of such whitespace in a response might be ignored by some clients or cause others to cease parsing.

3.1. Start Line

An HTTP message can be either a request from client to server or a response from server to client. Syntactically, the two types

3. メッセージフォーマット

すべてのHTTP / 1.1メッセージは、インターネットメッセージ形式[RFC5322]と同様の形式の開始行とそれに続く一連のオクテットで構成されます：0個以上のヘッダーフィールド（まとめて「ヘッダー」または「ヘッダーセクション」と呼ばれます）、ヘッダーセクションの終わりを示す空の行、およびオプションのメッセージ本文。

HTTP-message = start-line *（header-field CRLF）CRLF [message-body]

HTTPメッセージを解析する通常の手順は、開始行を構造体に読み込み、各ヘッダーフィールドをフィールド名でハッシュテーブルに読み込み、空の行まで読み込んだ後、解析されたデータを使用してメッセージ本文が予想されるかどうかを判断します。メッセージ本文が示されている場合は、メッセージ本文の長さに等しいオクテットの量が読み取られるか、接続が閉じられるまで、ストリームとして読み取られます。

受信者は、HTTPメッセージをUS-ASCII [USASCII]のスーパーセットであるエンコーディングのオクテットのシーケンスとして解析する必要があります。特定のエンコーディングに関係なく、HTTPメッセージをUnicode文字のストリームとして解析すると、文字列処理ライブラリがオクテットLF（%x0A）を含む無効なマルチバイト文字シーケンスを処理する方法が異なるため、セキュリティの脆弱性が生じます。文字列ベースのパーサーを安全に使用できるのは、メッセージの要素がメッセージから抽出された後（たとえば、メッセージの解析によって個々のフィールドが記述された後のヘッダーフィールド値内など）です。

HTTPメッセージは、増分処理またはダウンストリーム転送のためのストリームとして解析できます。ただし、一部の実装では、ネットワーク効率、セキュリティチェック、またはペイロード変換のためにメッセージの転送をバッファリングまたは遅延するため、受信者は部分的なメッセージの増分配信に依存できません。

送信者は、開始行と最初のヘッダーフィールドの間に空白を送信してはなりません（MUST NOT）。開始行と最初のヘッダーフィールドの間の空白を受信する受信者は、メッセージを無効として拒否するか、各空白行の前の行をそれ以上処理せずに消費する必要があります（つまり、行全体を無視し、その後に空白行が続く行を無視します）。 、適切に形成されたヘッダーフィールドが受信されるか、ヘッダーセクションが終了するまで）。

リクエスト内にこのような空白があると、サーバーをだましてそのフィールドを無視させたり、その行を新しいリクエストとして処理させたりする可能性があります。リクエストチェーン内の他の実装が同じことを解釈すると、セキュリティ上の脆弱性が発生する可能性があります。別のメッセージ。同様に、応答にこのような空白が含まれていると、一部のクライアントによって無視されたり、他のクライアントが解析を中止したりする場合があります。

3.1. スタートライン

HTTPメッセージは、クライアントからサーバーへの要求またはサーバーからクライアントへの応答のいずれかです。構文的に

of message differ only in the start-line, which is either a request-line (for requests) or a status-line (for responses), and in the algorithm for determining the length of the message body (Section 3.3).

In theory, a client could receive requests and a server could receive responses, distinguishing them by their different start-line formats, but, in practice, servers are implemented to only expect a request (a response is interpreted as an unknown or invalid request method) and clients are implemented to only expect a response.

start-line = request-line / status-line

3.1.1. Request Line

A request-line begins with a method token, followed by a single space (SP), the request-target, another single space (SP), the protocol version, and ends with CRLF.

request-line = method SP request-target SP HTTP-version CRLF

The method token indicates the request method to be performed on the target resource. The request method is case-sensitive.

method = token

The request methods defined by this specification can be found in Section 4 of [RFC7231], along with information regarding the HTTP method registry and considerations for defining new methods.

The request-target identifies the target resource upon which to apply the request, as defined in Section 5.3.

Recipients typically parse the request-line into its component parts by splitting on whitespace (see Section 3.5), since no whitespace is allowed in the three components. Unfortunately, some user agents fail to properly encode or exclude whitespace found in hypertext references, resulting in those disallowed characters being sent in a request-target.

Recipients of an invalid request-line SHOULD respond with either a 400 (Bad Request) error or a 301 (Moved Permanently) redirect with the request-target properly encoded. A recipient SHOULD NOT attempt to autocorrect and then process the request without a redirect, since the invalid request-line might be deliberately crafted to bypass security filters along the request chain.

HTTP does not place a predefined limit on the length of a request-line, as described in Section 2.5. A server that receives a method longer than any that it implements SHOULD respond with a 501 (Not Implemented) status code. A server that receives a request-target longer than any URI it wishes to parse MUST respond with a 414 (URI Too Long) status code (see Section 6.5.12 of [RFC7231]).

Various ad hoc limitations on request-line length are found in practice. It is RECOMMENDED that all HTTP senders and recipients support, at a minimum, request-line lengths of 8000 octets.

は、2種類のメッセージは、リクエストライン（リクエストの場合）またはステータスライン（レスポンスの場合）のスタートラインと、メッセージ本文の長さを決定するアルゴリズム（セクション3.3）。

理論的には、クライアントは要求を受信し、サーバーは応答を受信して、異なる開始行形式でそれらを区別できますが、実際には、サーバーは要求のみを予期するように実装されます（応答は不明または無効な要求メソッドとして解釈されます））とクライアントは、応答のみを期待するように実装されています。

start-line = request-line / status-line

3.1.1. リクエストライン

リクエストラインはメソッドトークンで始まり、シングルスペース（SP）、リクエストターゲット、別のシングルスペース（SP）、プロトコルバージョンが続き、CRLFで終わります。

request-line = method SP request-target SP HTTP-version CRLF

メソッドトークンは、ターゲットリソースで実行されるリクエストメソッドを示します。リクエストメソッドでは大文字と小文字が区別されます。

メソッド=トークン

この仕様で定義されているリクエストメソッドは、[RFC7231]のセクション4、HTTPメソッドレジストリに関する情報、および新しいメソッドを定義するための考慮事項に記載されています。

request-targetは、セクション5.3で定義されているように、リクエストを適用するターゲットリソースを識別します。

受信者は通常、リクエストラインを空白で分割することでコンポーネント部分に解析します（セクション3.5を参照）。これは、3つのコンポーネントでは空白を使用できないためです。残念ながら、一部のユーザーエージェントは、ハイパーテキスト参照にある空白を適切にエンコードまたは除外できず、その結果、許可されていない文字がリクエストターゲットで送信されます。

無効なリクエストラインの受信者は、リクエストターゲットが適切にエンコードされた400（Bad Request）エラーまたは301（Moved Permanently）リダイレクトのいずれかで応答する必要があります（SHOULD）。無効なrequest-lineはリクエストチェーンに沿ってセキュリティフィルターを迂回するように意図的に作成されている可能性があるため、受信者はリダイレクトを行わずにリクエストを自動修正してから処理しようとしないでください。

セクション2.5で説明されているように、HTTPではリクエストラインの長さに事前定義の制限はありません。実装するメソッドよりも長いメソッドを受信するサーバーは、501（未実装）ステータスコードで応答する必要があります（SHOULD）。解析するURIよりも長いリクエストターゲットを受信したサーバーは、414（URIが長すぎる）ステータスコードで応答する必要があります（[RFC7231]のセクション6.5.12を参照）。

リクエストラインの長さに関するさまざまなアドホック制限が実際に見られます。すべてのHTTP送信者と受信者が、少なくとも8000オクテットのリクエストライン長をサポートすることが推奨されます。

3.1.2. Status Line

The first line of a response message is the status-line, consisting of the protocol version, a space (SP), the status code, another space, a possibly empty textual phrase describing the status code, and ending with CRLF.

status-line = HTTP-version SP status-code SP reason-phrase CRLF

The status-code element is a 3-digit integer code describing the result of the server's attempt to understand and satisfy the client's corresponding request. The rest of the response message is to be interpreted in light of the semantics defined for that status code. See Section 6 of [RFC7231] for information about the semantics of status codes, including the classes of status code (indicated by the first digit), the status codes defined by this specification, considerations for the definition of new status codes, and the IANA registry.

status-code = 3DIGIT

The reason-phrase element exists for the sole purpose of providing a textual description associated with the numeric status code, mostly out of deference to earlier Internet application protocols that were more frequently used with interactive text clients. A client SHOULD ignore the reason-phrase content.

```
reason-phrase  = *( HTAB / SP / VCHAR / obs-text )
```

3.1.2. ステータスライン

応答メッセージの最初の行は、プロトコルバージョン、スペース（SP）、ステータスコード、別のスペース、ステータスコードを説明する空のテキストフレーズで構成され、CRLFで終わるステータス行です。

status-line = HTTP-version SP status-code SP reason-phrase CRLF

status-code要素は、サーバーがクライアントの対応する要求を理解して満足する試みの結果を説明する3桁の整数コードです。残りの応答メッセージは、そのステータスコードに定義されているセマンティクスに照らして解釈されます。ステータスコードのクラス（最初の桁で示される）、この仕様で定義されているステータスコード、新しいステータスコードの定義に関する考慮事項、IANAなど、ステータスコードのセマンティクスについては、[RFC7231]のセクション6を参照してください。レジストリ。

ステータスコード= 3DIGIT

reason-phrase要素は、数値のステータスコードに関連付けられたテキストによる説明を提供することを唯一の目的として存在します。これは主に、インタラクティブテキストクライアントでより頻繁に使用されていた以前のインターネットアプリケーションプロトコルを順守しないためです。クライアントは、理由フレーズの内容を無視する必要があります（SHOULD）。

```
reason-phrase  = *( HTAB / SP / VCHAR / obs-text )
```

3.2. Header Fields

Each header field consists of a case-insensitive field name followed by a colon (":"), optional leading whitespace, the field value, and optional trailing whitespace.

header-field = field-name ":" OWS field-value OWS

```
field-name     = token
field-value    = *( field-content / obs-fold )
field-content  = field-vchar [ 1*( SP / HTAB ) field-vchar ]
field-vchar    = VCHAR / obs-text

obs-fold       = CRLF 1*( SP / HTAB )
               ; obsolete line folding
               ; see Section 3.2.4
```

3.2. ヘッダーフィールド

各ヘッダーフィールドは、大文字と小文字を区別しないフィールド名と、その後に続くコロン（ "：" ）、オプションの先頭の空白、フィールド値、およびオプションの末尾の空白で構成されます。

header-field = field-name "：" OWS field-value OWS

```
field-name     = token
field-value    = *( field-content / obs-fold )
field-content  = field-vchar [ 1*( SP / HTAB ) field-vchar ]
field-vchar    = VCHAR / obs-text

obs-fold       = CRLF 1*( SP / HTAB )
               ; obsolete line folding
               ; see Section 3.2.4
```

The field-name token labels the corresponding field-value as having the semantics defined by that header field. For example, the Date header field is defined in Section 7.1.1.2 of [RFC7231] as containing the origination timestamp for the message in which it appears.

field-nameトークンは、対応するfield-valueに、そのヘッダーフィールドで定義されたセマンティクスを持つものとしてラベルを付けます。たとえば、Dateヘッダーフィールドは、[RFC7231]のセクション7.1.1.2で、それが表示されるメッセージの開始タイムスタンプを含むものとして定義されています。

3.2.1. Field Extensibility

Header fields are fully extensible: there is no limit on the introduction of new field names, each presumably defining new semantics, nor on the number of header fields used in a given message. Existing fields are defined in each part of this specification and in many other specifications outside this document set.

New header fields can be defined such that, when they are understood by a recipient, they might override or enhance the interpretation of previously defined header fields, define

3.2.1. フィールドの拡張性

ヘッダーフィールドは完全に拡張可能です。新しいフィールド名の導入、それぞれが新しいセマンティクスを定義すると思われるもの、または特定のメッセージで使用されるヘッダーフィールドの数に制限はありません。既存のフィールドは、この仕様の各部分およびこのドキュメントセット以外の多くの仕様で定義されています。

新しいヘッダーフィールドは、受信者が理解したときに、以前に定義されたヘッダーフィールドの解釈を上書きまたは強化したり、要求評価の前提条件を定義したり、応答の意味を調整し

preconditions on request evaluation, or refine the meaning of responses.

たりするように定義できます。

A proxy MUST forward unrecognized header fields unless the field-name is listed in the Connection header field (Section 6.1) or the proxy is specifically configured to block, or otherwise transform, such fields. Other recipients SHOULD ignore unrecognized header fields. These requirements allow HTTP's functionality to be enhanced without requiring prior update of deployed intermediaries.

プロキシは、フィールド名が接続ヘッダーフィールド（セクション6.1）にリストされていないか、プロキシがそのようなフィールドをブロックまたは変換するように特別に構成されていない限り、認識されないヘッダーフィールドを転送する必要があります。他の受信者は、認識されないヘッダーフィールドを無視する必要があります（SHOULD）。これらの要件により、展開された仲介者を事前に更新することなく、HTTPの機能を拡張できます。

All defined header fields ought to be registered with IANA in the "Message Headers" registry, as described in Section 8.3 of [RFC7231].

[RFC7231]のセクション8.3で説明されているように、定義されたすべてのヘッダーフィールドは、「メッセージヘッダー」レジストリのIANAに登録する必要があります。

3.2.2. Field Order

3.2.2. フィールドオーダー

The order in which header fields with differing field names are received is not significant. However, it is good practice to send header fields that contain control data first, such as Host on requests and Date on responses, so that implementations can decide when not to handle a message as early as possible. A server MUST NOT apply a request to the target resource until the entire request header section is received, since later header fields might include conditionals, authentication credentials, or deliberately misleading duplicate header fields that would impact request processing.

異なるフィールド名を持つヘッダーフィールドが受信される順序は重要ではありません。ただし、要求のホストや応答の日付など、制御データを含むヘッダーフィールドを最初に送信することをお勧めします。これにより、実装は、メッセージをできるだけ早く処理しない場合を決定できます。後のヘッダーフィールドに条件、認証資格情報、またはリクエスト処理に影響を与える意図的に誤解を招く重複ヘッダーフィールドが含まれる可能性があるため、サーバーはリクエストヘッダーセクション全体を受信するまで、ターゲットリソースにリクエストを適用してはなりません（MUST NOT）。

A sender MUST NOT generate multiple header fields with the same field name in a message unless either the entire field value for that header field is defined as a comma-separated list [i.e., #(values)] or the header field is a well-known exception (as noted below).

送信者は、そのヘッダーフィールドのフィールド値全体がコンマ区切りリストとして定義されている場合（つまり、 #（values）]またはヘッダーフィールドが適切な場合を除き、メッセージ内に同じフィールド名を持つ複数のヘッダーフィールドを生成してはなりません（MUST NOT）。既知の例外（下記のとおり）。

A recipient MAY combine multiple header fields with the same field name into one "field-name: field-value" pair, without changing the semantics of the message, by appending each subsequent field value to the combined field value in order, separated by a comma. The order in which header fields with the same field name are received is therefore significant to the interpretation of the combined field value; a proxy MUST NOT change the order of these field values when forwarding a message.

受信者は、メッセージのセマンティクスを変更せずに、同じフィールド名を持つ複数のヘッダーフィールドを1つの「フィールド名：フィールド値」ペアに結合することができます。コンマ。したがって、同じフィールド名を持つヘッダーフィールドが受信される順序は、結合されたフィールド値の解釈にとって重要です。プロキシは、メッセージを転送するときにこれらのフィールド値の順序を変更してはなりません（MUST NOT）。

Note: In practice, the "Set-Cookie" header field ([RFC6265]) often appears multiple times in a response message and does not use the list syntax, violating the above requirements on multiple header fields with the same name. Since it cannot be combined into a single field-value, recipients ought to handle "Set-Cookie" as a special case while processing header fields. (See Appendix A.2.3 of [Kri2001] for details.)

注：実際には、 "Set-Cookie"ヘッダーフィールド（[RFC6265]）は応答メッセージに複数回表示されることが多く、リスト構文を使用していないため、同じ名前の複数のヘッダーフィールドに対する上記の要件に違反しています。単一のフィールド値に組み合わせることができないため、受信者はヘッダーフィールドの処理中に「Set-Cookie」を特殊なケースとして処理する必要があります。 （詳細については、[Kri2001]の付録A.2.3を参照してください。）

3.2.3. Whitespace

3.2.3. 空白

This specification uses three rules to denote the use of linear whitespace: OWS (optional whitespace), RWS (required whitespace), and BWS ("bad" whitespace).

この仕様では、線形空白の使用を示すために、OWS（オプションの空白）、RWS（必須の空白）、およびBWS（「不正な」空白）の3つのルールを使用しています。

The OWS rule is used where zero or more linear whitespace octets might appear. For protocol elements where optional whitespace is preferred to improve readability, a sender SHOULD generate the optional whitespace as a single SP;

OWSルールは、ゼロ個以上の線形空白オクテットが出現する可能性がある場合に使用されます。読みやすさを向上させるためにオプションの空白が推奨されるプロトコル要素の場合、送信者はオプションの空白を単一のSPとして生成する必要があります（SHOULD）。それ以外の場合、送信者は、インプレースメ

otherwise, a sender SHOULD NOT generate optional whitespace except as needed to white out invalid or unwanted protocol elements during in-place message filtering.

ッセージフィルタリング中に無効または不要なプロトコル要素をホワイトアウトする必要がある場合を除いて、オプションの空白を生成してはなりません（SHOULD NOT）。

The RWS rule is used when at least one linear whitespace octet is required to separate field tokens. A sender SHOULD generate RWS as a single SP.

RWSルールは、フィールドトークンを区切るために少なくとも1つの線形空白オクテットが必要な場合に使用されます。送信者はRWSを単一のSPとして生成する必要があります（SHOULD）。

The BWS rule is used where the grammar allows optional whitespace only for historical reasons. A sender MUST NOT generate BWS in messages. A recipient MUST parse for such bad whitespace and remove it before interpreting the protocol element.

BWSルールは、文法が歴史的な理由でのみオプションの空白を許可する場合に使用されます。送信者はメッセージでBWSを生成してはいけません（MUST NOT）。受信者は、そのような不正な空白を解析して、プロトコル要素を解釈する前に削除する必要があります。

```
    OWS            = *( SP / HTAB )
                   ; optional whitespace
    RWS            = 1*( SP / HTAB )
                   ; required whitespace
    BWS            = OWS
                   ; "bad" whitespace
```

3.2.4. Field Parsing

3.2.4. フィールド解析

Messages are parsed using a generic algorithm, independent of the individual header field names. The contents within a given field value are not parsed until a later stage of message interpretation (usually after the message's entire header section has been processed). Consequently, this specification does not use ABNF rules to define each "Field-Name: Field Value" pair, as was done in previous editions. Instead, this specification uses ABNF rules that are named according to each registered field name, wherein the rule defines the valid grammar for that field's corresponding field values (i.e., after the field-value has been extracted from the header section by a generic field parser).

メッセージは、個々のヘッダーフィールド名に関係なく、一般的なアルゴリズムを使用して解析されます。特定のフィールド値の内容は、メッセージ解釈の後の段階（通常はメッセージのヘッダーセクション全体が処理された後）まで解析されません。したがって、この仕様では、以前の版で行われていたように、ABNFルールを使用して各「フィールド名：フィールド値」のペアを定義していません。代わりに、この仕様は、登録された各フィールド名に従って名前が付けられたABNFルールを使用します。このルールは、そのフィールドの対応するフィールド値の有効な文法を定義します（つまり、フィールド値が汎用フィールドパーサーによってヘッダーセクションから抽出された後）））。

No whitespace is allowed between the header field-name and colon. In the past, differences in the handling of such whitespace have led to security vulnerabilities in request routing and response handling. A server MUST reject any received request message that contains whitespace between a header field-name and colon with a response code of 400 (Bad Request). A proxy MUST remove any such whitespace from a response message before forwarding the message downstream.

ヘッダーのフィールド名とコロンの間に空白を入れることはできません。以前は、このような空白の処理の違いにより、要求のルーティングと応答の処理にセキュリティ上の脆弱性がありました。サーバーは、ヘッダーフィールド名とコロンの間に空白を含み、応答コード400（Bad Request）で受信した要求メッセージを拒否する必要があります。プロキシは、メッセージをダウンストリームに転送する前に、応答メッセージからそのような空白を削除する必要があります。

A field value might be preceded and/or followed by optional whitespace (OWS); a single SP preceding the field-value is preferred for consistent readability by humans. The field value does not include any leading or trailing whitespace: OWS occurring before the first non-whitespace octet of the field value or after the last non-whitespace octet of the field value ought to be excluded by parsers when extracting the field value from a header field.

フィールド値の前後には、オプションの空白（OWS）が続く場合があります。人間が一貫して読みやすくするために、フィールド値の前の単一のSPが推奨されます。フィールド値には、先頭または末尾の空白は含まれません。フィールド値の最初の非空白オクテットの前、またはフィールド値の最後の非空白オクテットの後に発生するOWSは、フィールド値をフィールドから抽出するときにパーサーによって除外されます。ヘッダーフィールド。

Historically, HTTP header field values could be extended over multiple lines by preceding each extra line with at least one space or horizontal tab (obs-fold). This specification deprecates such line folding except within the message/http media type (Section 8.3.1). A sender MUST NOT generate a message that includes line folding (i.e., that has any field-value that contains a match to the obs-fold rule) unless the message is intended for packaging within the message/http media type.

従来、HTTPヘッダーフィールドの値は、余分な各行の前に少なくとも1つのスペースまたは水平タブ（obs-fold）を付けることで、複数行に拡張できました。この仕様では、message / httpメディアタイプ（セクション8.3.1）内を除き、このような行の折りたたみを廃止しています。送信者は、メッセージがメッセージ/ httpメディアタイプ内でのパッケージ化を意図していない限り、行の折りたたみを含むメッセージ（つまり、obs-foldルールへの一致を含むフィールド値を持つメッセージ）を生成してはなりません（MUST NOT）。

A server that receives an obs-fold in a request message that is not within a message/http container MUST either reject the message by sending a 400 (Bad Request), preferably with a representation explaining that obsolete line folding is unacceptable, or replace each received obs-fold with one or more SP octets prior to interpreting the field value or forwarding the message downstream.

メッセージ/ httpコンテナー内にない要求メッセージでobs-foldを受信するサーバーは、400（Bad Request）を送信してメッセージを拒否する必要があります（古い行の折りたたみは許容できないことを説明する表現を使用するか、それぞれを置き換える必要があります）。フィールド値を解釈するか、メッセージをダウンストリームに転送する前に、1つ以上のSPオクテットでobs-foldを受信しました。

A proxy or gateway that receives an obs-fold in a response message that is not within a message/http container MUST either discard the message and replace it with a 502 (Bad Gateway) response, preferably with a representation explaining that unacceptable line folding was received, or replace each received obs-fold with one or more SP octets prior to interpreting the field value or forwarding the message downstream.

メッセージ/ httpコンテナ内にない応答メッセージでobs-foldを受信するプロキシまたはゲートウェイは、メッセージを破棄し、それを502（不正なゲートウェイ）応答に置き換える必要があります。フィールド値を解釈するか、メッセージをダウンストリームに転送する前に、受信するか、受信した各obs-foldを1つ以上のSPオクテットに置き換えます。

A user agent that receives an obs-fold in a response message that is not within a message/http container MUST replace each received obs-fold with one or more SP octets prior to interpreting the field value.

メッセージ/ httpコンテナー内にない応答メッセージでobs-foldを受信するユーザーエージェントは、フィールド値を解釈する前に、受信した各obs-foldを1つ以上のSPオクテットに置き換えなければなりません（MUST）。

Historically, HTTP has allowed field content with text in the ISO-8859-1 charset [ISO-8859-1], supporting other charsets only through use of [RFC2047] encoding. In practice, most HTTP header field values use only a subset of the US-ASCII charset [USASCII]. Newly defined header fields SHOULD limit their field values to US-ASCII octets. A recipient SHOULD treat other octets in field content (obs-text) as opaque data.

歴史的に、HTTPはISO-8859-1文字セット[ISO-8859-1]のテキストを含むフィールドコンテンツを許可しており、[RFC2047]エンコーディングの使用を通じてのみ他の文字セットをサポートしていました。実際には、ほとんどのHTTPヘッダーフィールド値はUS-ASCII文字セット[USASCII]のサブセットのみを使用します。新しく定義されたヘッダーフィールドは、フィールド値をUS-ASCIIオクテットに制限する必要があります（SHOULD）。受信者は、フィールドコンテンツ（obs-text）の他のオクテットを不透明なデータとして扱う必要があります（SHOULD）。

3.2.5. Field Limits

HTTP does not place a predefined limit on the length of each header field or on the length of the header section as a whole, as described in Section 2.5. Various ad hoc limitations on individual header field length are found in practice, often depending on the specific field semantics.

3.2.5. フィールド制限

セクション2.5で説明されているように、HTTPは、各ヘッダーフィールドの長さやヘッダーセクション全体の長さを事前に定義していません。個々のヘッダーフィールド長に関するさまざまなアドホック制限が実際に見られますが、多くの場合、特定のフィールドセマンティクスによって異なります。

A server that receives a request header field, or set of fields, larger than it wishes to process MUST respond with an appropriate 4xx (Client Error) status code. Ignoring such header fields would increase the server's vulnerability to request smuggling attacks (Section 9.5).

処理したいサイズより大きいリクエストヘッダーフィールドまたはフィールドセットを受信するサーバーは、適切な4xx（クライアントエラー）ステータスコードで応答する必要があります。このようなヘッダーフィールドを無視すると、サーバーの脆弱性が高まり、密輸攻撃をリクエストするようになります（セクション9.5）。

A client MAY discard or truncate received header fields that are larger than the client wishes to process if the field semantics are such that the dropped value(s) can be safely ignored without changing the message framing or response semantics.

クライアントは、フィールドのセマンティクスがメッセージのフレーミングまたは応答のセマンティクスを変更せずにドロップされた値を安全に無視できるようなものである場合、クライアントが処理したいより大きい受信ヘッダーフィールドを破棄または切り捨てることができます。

3.2.6. Field Value Components

Most HTTP header field values are defined using common syntax components (token, quoted-string, and comment) separated by whitespace or specific delimiting characters. Delimiters are chosen from the set of US-ASCII visual characters not allowed in a token (DQUOTE and "(),/:;<=>?@[\]{}").

3.2.6. フィールド値コンポーネント

ほとんどのHTTPヘッダーフィールド値は、空白または特定の区切り文字で区切られた一般的な構文コンポーネント（トークン、引用符付き文字列、コメント）を使用して定義されます。区切り文字は、トークンで許可されないUS-ASCIIビジュアル文字のセットから選択されます（DQUOTEおよび "()、/ :; <=>？@ [\] {}"）。

token = 1*tchar

トークン= 1 * tchar

```
    tchar          = "!" / "#" / "$" / "%" / "&" / "'" / "*"
                   / "+" / "-" / "." / "^" / "_" / "`" / "|" / "~"
```

```
                / DIGIT / ALPHA
                ; any VCHAR, except delimiters
```

A string of text is parsed as a single value if it is quoted using double-quote marks.

テキストの文字列は、二重引用符で囲まれている場合、単一の値として解析されます。

```
    quoted-string  = DQUOTE *( qdtext / quoted-pair ) DQUOTE
    qdtext         = HTAB / SP /%x21 / %x23-5B / %x5D-7E / obs-text
    obs-text       = %x80-FF
```

Comments can be included in some HTTP header fields by surrounding the comment text with parentheses. Comments are only allowed in fields containing "comment" as part of their field value definition.

コメントは、コメントテキストを括弧で囲むことにより、一部のHTTPヘッダーフィールドに含めることができます。コメントは、フィールド値の定義の一部として「コメント」を含むフィールドでのみ許可されます。

```
    comment        = "(" *( ctext / quoted-pair / comment ) ")"
    ctext          = HTAB / SP / %x21-27 / %x2A-5B / %x5D-7E / obs-text
```

The backslash octet ("\") can be used as a single-octet quoting mechanism within quoted-string and comment constructs. Recipients that process the value of a quoted-string MUST handle a quoted-pair as if it were replaced by the octet following the backslash.

バックスラッシュオクテット（ "\"）は、引用文字列およびコメント構造内の単一オクテット引用メカニズムとして使用できます。 quoted-stringの値を処理する受信者は、引用符で囲まれたペアを、バックスラッシュに続くオクテットに置き換えられたかのように処理する必要があります。

```
    quoted-pair    = "\" ( HTAB / SP / VCHAR / obs-text )
```

A sender SHOULD NOT generate a quoted-pair in a quoted-string except where necessary to quote DQUOTE and backslash octets occurring within that string. A sender SHOULD NOT generate a quoted-pair in a comment except where necessary to quote parentheses ["(" and ")"] and backslash octets occurring within that comment.

送信者は、DQUOTEおよびその文字列内で発生するバックスラッシュオクテットを引用する必要がある場合を除いて、引用文字列で引用ペアを生成してはなりません（SHOULD NOT）。送信者は、コメント内で発生する括弧["（"と "）"]とバックスラッシュオクテットを引用する必要がある場合を除いて、コメントに引用符で囲まれたペアを生成してはなりません（SHOULD NOT）。

3.3. Message Body

3.3. メッセージ本文

The message body (if any) of an HTTP message is used to carry the payload body of that request or response. The message body is identical to the payload body unless a transfer coding has been applied, as described in Section 3.3.1.

HTTPメッセージのメッセージ本文（存在する場合）は、その要求または応答のペイロード本文を運ぶために使用されます。セクション3.3.1で説明されているように、転送コーディングが適用されていない限り、メッセージ本文はペイロード本文と同じです。

```
    message-body = *OCTET
```

The rules for when a message body is allowed in a message differ for requests and responses.

メッセージでメッセージ本文を使用できる場合のルールは、要求と応答で異なります。

The presence of a message body in a request is signaled by a Content-Length or Transfer-Encoding header field. Request message framing is independent of method semantics, even if the method does not define any use for a message body.

リクエスト内のメッセージ本文の存在は、Content-LengthまたはTransfer-Encodingヘッダーフィールドによって通知されます。メソッドがメッセージ本文の使用を定義していない場合でも、要求メッセージのフレーミングはメソッドのセマンティクスとは無関係です。

The presence of a message body in a response depends on both the request method to which it is responding and the response status code (Section 3.1.2). Responses to the HEAD request method (Section 4.3.2 of [RFC7231]) never include a message body because the associated response header fields (e.g., Transfer-Encoding, Content-Length, etc.), if present, indicate only what their values would have been if the request method had been GET (Section 4.3.1 of [RFC7231]). 2xx (Successful) responses to a CONNECT request method (Section 4.3.6 of [RFC7231]) switch to tunnel mode instead of having a message body. All 1xx (Informational), 204 (No

応答内のメッセージ本文の存在は、応答先のリクエストメソッドと応答ステータスコードの両方に依存します（セクション3.1.2）。 HEADリクエストメソッドへの応答（[RFC7231]のセクション4.3.2）には、メッセージ本文が含まれることはありません。これは、関連する応答ヘッダーフィールド（Transfer-Encoding、Content-Lengthなど）が存在する場合、それらの値のみを示すためです。リクエストメソッドがGETであった場合（[RFC7231]のセクション4.3.1）。 CONNECTリクエストメソッド（[RFC7231]のセクション4.3.6）への2xx（成功）応答は、メッセージ本文ではなくトンネルモードに切り替わります。すべての1xx（情報）、204（コンテンツなし）、および304（変

Content), and 304 (Not Modified) responses do not include a message body. All other responses do include a message body, although the body might be of zero length.

更なし）応答には、メッセージ本文は含まれません。他のすべての応答にはメッセージ本文が含まれますが、本文の長さがゼロの場合もあります。

3.3.1. Transfer-Encoding

The Transfer-Encoding header field lists the transfer coding names corresponding to the sequence of transfer codings that have been (or will be) applied to the payload body in order to form the message body. Transfer codings are defined in Section 4.

```
Transfer-Encoding = 1#transfer-coding
```

Transfer-Encoding is analogous to the Content-Transfer-Encoding field of MIME, which was designed to enable safe transport of binary data over a 7-bit transport service ([RFC2045], Section 6). However, safe transport has a different focus for an 8bit-clean transfer protocol. In HTTP's case, Transfer-Encoding is primarily intended to accurately delimit a dynamically generated payload and to distinguish payload encodings that are only applied for transport efficiency or security from those that are characteristics of the selected resource.

A recipient MUST be able to parse the chunked transfer coding (Section 4.1) because it plays a crucial role in framing messages when the payload body size is not known in advance. A sender MUST NOT apply chunked more than once to a message body (i.e., chunking an already chunked message is not allowed). If any transfer coding other than chunked is applied to a request payload body, the sender MUST apply chunked as the final transfer coding to ensure that the message is properly framed. If any transfer coding other than chunked is applied to a response payload body, the sender MUST either apply chunked as the final transfer coding or terminate the message by closing the connection.

For example,

```
Transfer-Encoding: gzip, chunked
```

indicates that the payload body has been compressed using the gzip coding and then chunked using the chunked coding while forming the message body.

Unlike Content-Encoding (Section 3.1.2.1 of [RFC7231]), Transfer-Encoding is a property of the message, not of the representation, and any recipient along the request/response chain MAY decode the received transfer coding(s) or apply additional transfer coding(s) to the message body, assuming that corresponding changes are made to the Transfer-Encoding field-value. Additional information about the encoding parameters can be provided by other header fields not defined by this specification.

Transfer-Encoding MAY be sent in a response to a HEAD request or in a 304 (Not Modified) response (Section 4.1 of [RFC7232]) to a GET request, neither of which includes a message body, to indicate that the origin server would have applied a transfer coding to the message body if the request had been an unconditional GET. This indication is not required, however, because any recipient on the response chain (including the origin server) can remove transfer codings when they are not needed.

3.3.1. 転送エンコーディング

Transfer-Encodingヘッダーフィールドには、メッセージ本文を形成するためにペイロード本文に適用された（または今後適用される）転送コーディングのシーケンスに対応する転送コーディング名が一覧表示されます。転送コーディングはセクション4で定義されています。

```
Transfer-Encoding = 1 # transfer-coding
```

Transfer-Encodingは、MIMEのContent-Transfer-Encodingフィールドに似ています。これは、7ビットのトランスポートサービス（[RFC2045]、セクション6）を介してバイナリデータを安全にトランスポートできるように設計されています。ただし、安全な転送では、8ビットのクリーンな転送プロトコルに対して別の焦点があります。 HTTPの場合、Transfer-Encodingは主に、動的に生成されたペイロードを正確に区切り、転送効率またはセキュリティのみに適用されるペイロードエンコーディングを、選択されたリソースの特性と区別することを目的としています。

受信者は、チャンク転送コーディング（セクション4.1）を解析できなければなりません（ペイロードボディのサイズが事前にわからない場合、メッセージのフレーミングに重要な役割を果たすため）。送信者は、チャンクをメッセージ本文に2回以上適用してはなりません（つまり、すでにチャンクされたメッセージをチャンクすることは許可されていません）。チャンク以外の転送コーディングが要求ペイロード本体に適用される場合、送信者はメッセージが適切にフレーム化されることを保証するために、チャンクを最終転送コーディングとして適用する必要があります。チャンク以外の転送コーディングが応答ペイロード本体に適用される場合、送信者はチャンクを最終的な転送コーディングとして適用するか、接続を閉じることによってメッセージを終了する必要があります。

例えば、

```
Transfer-Encoding：gzip、チャンク
```

ペイロードボディがgzipコーディングを使用して圧縮され、メッセージボディの形成中にチャンクコーディングを使用してチャンク化されたことを示します。

Content-Encoding（[RFC7231]のセクション3.1.2.1）とは異なり、Transfer-Encodingは表現ではなくメッセージのプロパティであり、要求/応答チェーンに沿った受信者は、受信した転送コーディングをデコードまたは適用できます（MAY）。 Transfer-Encodingフィールド値に対応する変更が行われたと想定して、メッセージ本文に追加の転送コーディングを追加します。エンコーディングパラメータに関する追加情報は、この仕様で定義されていない他のヘッダーフィールドから提供できます。

Transfer-Encodingは、HEADリクエストへの応答、またはGETリクエストへの304（Not Modified）レスポンス（[RFC7232]のセクション4.1）で送信できますが、どちらにもメッセージ本文は含まれていません。リクエストが無条件のGETであった場合、メッセージ本文に転送コーディングを適用しました。ただし、応答チェーン（オリジンサーバーを含む）の受信者は、不要な転送コーディングを削除できるため、この指示は必要ありません。

A server MUST NOT send a Transfer-Encoding header field in any response with a status code of 1xx (Informational) or 204 (No Content). A server MUST NOT send a Transfer-Encoding header field in any 2xx (Successful) response to a CONNECT request (Section 4.3.6 of [RFC7231]).

サーバーは、ステータスコード1xx（情報）または204（コンテンツなし）の応答でTransfer-Encodingヘッダーフィールドを送信してはなりません（MUST NOT）。サーバーは、CONNECTリクエストへの2xx（成功）応答でTransfer-Encodingヘッダーフィールドを送信してはなりません（[RFC7231]のセクション4.3.6）。

Transfer-Encoding was added in HTTP/1.1. It is generally assumed that implementations advertising only HTTP/1.0 support will not understand how to process a transfer-encoded payload. A client MUST NOT send a request containing Transfer-Encoding unless it knows the server will handle HTTP/1.1 (or later) requests; such knowledge might be in the form of specific user configuration or by remembering the version of a prior received response. A server MUST NOT send a response containing Transfer-Encoding unless the corresponding request indicates HTTP/1.1 (or later).

Transfer-EncodingはHTTP / 1.1で追加されました。一般的に、HTTP / 1.0サポートのみをアドバタイズする実装は、転送エンコードされたペイロードの処理方法を理解しないと想定されています。クライアントが、サーバーがHTTP / 1.1（またはそれ以降）の要求を処理することを認識していない限り、Transfer-Encodingを含む要求を送信してはなりません（MUST NOT）。そのような知識は、特定のユーザー構成の形で、または以前に受け取った応答のバージョンを記憶することである場合があります。対応する要求がHTTP / 1.1（またはそれ以降）を示さない限り、サーバーはTransfer-Encodingを含む応答を送信してはなりません（MUST NOT）。

A server that receives a request message with a transfer coding it does not understand SHOULD respond with 501 (Not Implemented).

転送コーディングを含む要求メッセージを受信したサーバーは、501（実装されていません）で応答する必要がある（SHOULD）と理解しません。

3.3.2. Content-Length

3.3.2. コンテンツの長さ

When a message does not have a Transfer-Encoding header field, a Content-Length header field can provide the anticipated size, as a decimal number of octets, for a potential payload body. For messages that do include a payload body, the Content-Length field-value provides the framing information necessary for determining where the body (and message) ends. For messages that do not include a payload body, the Content-Length indicates the size of the selected representation (Section 3 of [RFC7231]).

メッセージにTransfer-Encodingヘッダーフィールドがない場合、Content-Lengthヘッダーフィールドは、潜在的なペイロード本体の予想サイズを10進数のオクテットとして提供できます。ペイロード本文を含むメッセージの場合、Content-Lengthフィールド値は、本文（およびメッセージ）の終了位置を決定するために必要なフレーミング情報を提供します。ペイロード本文を含まないメッセージの場合、Content-Lengthは選択された表現のサイズを示します（[RFC7231]のセクション3）。

```
Content-Length = 1*DIGIT
```

```
コンテンツの長さ= 1 * DIGIT
```

An example is

例は

```
Content-Length: 3495
```

```
コンテンツの長さ：3495
```

A sender MUST NOT send a Content-Length header field in any message that contains a Transfer-Encoding header field.

送信者は、Transfer-Encodingヘッダーフィールドを含むメッセージでContent-Lengthヘッダーフィールドを送信してはなりません（MUST NOT）。

A user agent SHOULD send a Content-Length in a request message when no Transfer-Encoding is sent and the request method defines a meaning for an enclosed payload body. For example, a Content-Length header field is normally sent in a POST request even when the value is 0 (indicating an empty payload body). A user agent SHOULD NOT send a Content-Length header field when the request message does not contain a payload body and the method semantics do not anticipate such a body.

ユーザーエージェントは、Transfer-Encodingが送信されず、リクエストメソッドが囲まれたペイロード本文の意味を定義している場合に、リクエストメッセージでContent-Lengthを送信する必要があります（SHOULD）。たとえば、Content-Lengthヘッダーフィールドは通常、値が0（空のペイロード本文を示す）の場合でも、POSTリクエストで送信されます。リクエストメッセージにペイロードボディが含まれておらず、メソッドセマンティクスがそのようなボディを予期していない場合、ユーザーエージェントはContent-Lengthヘッダーフィールドを送信しないでください。

A server MAY send a Content-Length header field in a response to a HEAD request (Section 4.3.2 of [RFC7231]); a server MUST NOT send Content-Length in such a response unless its field-value equals the decimal number of octets that would have been sent in the payload body of a response if the same request had used the GET method.

サーバーは、HEADリクエストへの応答でContent-Lengthヘッダーフィールドを送信できます（[RFC7231]のセクション4.3.2）。同じリクエストでGETメソッドを使用した場合に、フィールド値が応答のペイロード本体で送信された10進数のオクテットと等しい場合を除き、サーバーはそのような応答でContent-Lengthを送信してはなりません（MUST NOT）。

A server MAY send a Content-Length header field in a 304 (Not Modified) response to a conditional GET request (Section 4.1 of [RFC7232]); a server MUST NOT send Content-Length in such

サーバーは、条件付きGETリクエストへの304（Not Modified）レスポンスでContent-Lengthヘッダーフィールドを送信できます（[RFC7232]のセクション4.1）。サーバーはそのような応答

a response

でContent-Lengthを送信してはいけません（MUST NOT）

unless its field-value equals the decimal number of octets that would have been sent in the payload body of a 200 (OK) response to the same request.

そのフィールド値が、同じリクエストに対する200（OK）応答のペイロード本体で送信される10進数のオクテットと等しくない限り。

A server MUST NOT send a Content-Length header field in any response with a status code of 1xx (Informational) or 204 (No Content). A server MUST NOT send a Content-Length header field in any 2xx (Successful) response to a CONNECT request (Section 4.3.6 of [RFC7231]).

サーバーは、ステータスコード1xx（情報）または204（コンテンツなし）の応答でContent-Lengthヘッダーフィールドを送信してはなりません（MUST NOT）。サーバーは、CONNECTリクエストに対する2xx（成功）応答でContent-Lengthヘッダーフィールドを送信してはなりません（[RFC7231]のセクション4.3.6）。

Aside from the cases defined above, in the absence of Transfer-Encoding, an origin server SHOULD send a Content-Length header field when the payload body size is known prior to sending the complete header section. This will allow downstream recipients to measure transfer progress, know when a received message is complete, and potentially reuse the connection for additional requests.

上記で定義されたケースとは別に、Transfer-Encodingがない場合、完全なヘッダーセクションを送信する前にペイロード本体のサイズがわかっている場合、オリジンサーバーはContent-Lengthヘッダーフィールドを送信する必要があります（SHOULD）。これにより、ダウンストリームの受信者は転送の進行状況を測定し、受信したメッセージがいつ完了したかを把握し、接続を再利用して追加のリクエストを行うことができます。

Any Content-Length field value greater than or equal to zero is valid. Since there is no predefined limit to the length of a payload, a recipient MUST anticipate potentially large decimal numerals and prevent parsing errors due to integer conversion overflows (Section 9.3).

ゼロ以上のContent-Lengthフィールド値はすべて有効です。ペイロードの長さには事前定義の制限がないため、受信者は、潜在的に大きな10進数を予測し、整数変換オーバーフローによる解析エラーを防止する必要があります（9.3）。

If a message is received that has multiple Content-Length header fields with field-values consisting of the same decimal value, or a single Content-Length header field with a field value containing a list of identical decimal values (e.g., "Content-Length: 42, 42"), indicating that duplicate Content-Length header fields have been generated or combined by an upstream message processor, then the recipient MUST either reject the message as invalid or replace the duplicated field-values with a single valid Content-Length field containing that decimal value prior to determining the message body length or forwarding the message.

同じ10進数値で構成されるフィールド値を持つ複数のContent-Lengthヘッダーフィールドがあるメッセージ、または同一の10進数値のリストを含むフィールド値を持つ単一のContent-Lengthヘッダーフィールド（例：「Content-Length ：42、42 "）、重複したContent-Lengthヘッダーフィールドが上流のメッセージプロセッサによって生成または結合されたことを示し、受信者はメッセージを無効として拒否するか、重複したフィールド値を単一の有効なContent-Lengthで置き換える必要があります。メッセージ本文の長さを決定するか、メッセージを転送する前に、その10進数値を含むフィールド。

Note: HTTP's use of Content-Length for message framing differs significantly from the same field's use in MIME, where it is an optional field used only within the "message/external-body" media-type.

注：HTTPがメッセージのフレーミングにContent-Lengthを使用することは、MIMEでの同じフィールドの使用とは大きく異なります。このフィールドは、「message / external-body」メディアタイプ内でのみ使用されるオプションのフィールドです。

3.3.3. Message Body Length

3.3.3. メッセージ本文の長さ

The length of a message body is determined by one of the following (in order of precedence):

メッセージ本文の長さは、次のいずれかによって決定されます（優先順位順）。

1. Any response to a HEAD request and any response with a 1xx (Informational), 204 (No Content), or 304 (Not Modified) status code is always terminated by the first empty line after the header fields, regardless of the header fields present in the message, and thus cannot contain a message body.

1. HEADリクエストへの応答と、1xx（情報）、204（コンテンツなし）、または304（変更なし）ステータスコードの応答は、ヘッダーフィールドの存在に関係なく、常にヘッダーフィールドの後の最初の空行で終了します。メッセージであるため、メッセージ本文を含めることはできません。

2. Any 2xx (Successful) response to a CONNECT request implies that the connection will become a tunnel immediately after the empty line that concludes the header fields. A client MUST ignore any Content-Length or Transfer-Encoding header fields received in such a message.

2. CONNECT要求に対する2xx（成功）応答は、ヘッダーフィールドを終了する空の行の直後に接続がトンネルになることを意味します。クライアントは、そのようなメッセージで受信されたContent-LengthまたはTransfer-Encodingヘッダーフィールドを無視する必要があります。

3. If a Transfer-Encoding header field is present and the chunked transfer coding (Section 4.1) is the final encoding, the message body length is determined by reading and decoding

3. Transfer-Encodingヘッダーフィールドが存在し、チャンク転送コーディング（セクション4.1）が最終的なエンコーディングである場合、メッセージ本体の長さは、転送コーディングがデ

the chunked data until the transfer coding indicates the data is complete.

一タの完了を示すまでチャンクデータを読み取ってデコードすることによって決定されます。

If a Transfer-Encoding header field is present in a response and the chunked transfer coding is not the final encoding, the message body length is determined by reading the connection until it is closed by the server. If a Transfer-Encoding header field is present in a request and the chunked transfer coding is not the final encoding, the message body length cannot be determined reliably; the server MUST respond with the 400 (Bad Request) status code and then close the connection.

Transfer-Encodingヘッダーフィールドが応答に存在し、チャンク転送コーディングが最終的なエンコーディングでない場合、メッセージ本文の長さは、サーバーによって閉じられるまで接続を読み取ることによって決定されます。 Transfer-Encodingヘッダーフィールドがリクエストに存在し、チャンク転送コーディングが最終的なエンコーディングでない場合、メッセージ本文の長さを確実に決定できません。サーバーは400（Bad Request）ステータスコードで応答し、接続を閉じる必要があります。

If a message is received with both a Transfer-Encoding and a Content-Length header field, the Transfer-Encoding overrides the Content-Length. Such a message might indicate an attempt to perform request smuggling (Section 9.5) or response splitting (Section 9.4) and ought to be handled as an error. A sender MUST remove the received Content-Length field prior to forwarding such a message downstream.

Transfer-EncodingとContent-Lengthヘッダーフィールドの両方を含むメッセージを受信した場合、Transfer-EncodingはContent-Lengthを上書きします。このようなメッセージは、リクエストの密輸（セクション9.5）またはレスポンスの分割（セクション9.4）を実行しようとする試みを示している可能性があり、エラーとして処理する必要があります。送信者は、そのようなメッセージをダウンストリームに転送する前に、受信したContent-Lengthフィールドを削除する必要があります。

4. If a message is received without Transfer-Encoding and with either multiple Content-Length header fields having differing field-values or a single Content-Length header field having an invalid value, then the message framing is invalid and the recipient MUST treat it as an unrecoverable error. If this is a request message, the server MUST respond with a 400 (Bad Request) status code and then close the connection. If this is a response message received by a proxy, the proxy MUST close the connection to the server, discard the received response, and send a 502 (Bad Gateway) response to the client. If this is a response message received by a user agent, the user agent MUST close the connection to the server and discard the received response.

4. Transfer-Encodingなしで、異なるフィールド値を持つ複数のContent-Lengthヘッダーフィールドまたは無効な値を持つ単一のContent-Lengthヘッダーフィールドを持つメッセージを受信した場合、メッセージのフレーミングは無効であり、受信者はそれを回復不可能なエラー。これが要求メッセージである場合、サーバーは400（Bad Request）ステータスコードで応答してから、接続を閉じる必要があります。これがプロキシによって受信された応答メッセージである場合、プロキシはサーバーへの接続を閉じ、受信した応答を破棄し、502（不正なゲートウェイ）応答をクライアントに送信する必要があります。これがユーザーエージェントによって受信された応答メッセージである場合、ユーザーエージェントはサーバーへの接続を閉じ、受信した応答を破棄する必要があります。

5. If a valid Content-Length header field is present without Transfer-Encoding, its decimal value defines the expected message body length in octets. If the sender closes the connection or the recipient times out before the indicated number of octets are received, the recipient MUST consider the message to be incomplete and close the connection.

5. Transfer-Encodingなしで有効なContent-Lengthヘッダーフィールドが存在する場合、その10進数値により、予想されるメッセージ本文の長さがオクテットで定義されます。送信者が接続を閉じるか、指定された数のオクテットを受信する前に受信者がタイムアウトした場合、受信者はメッセージが不完全であると見なして接続を閉じる必要があります。

6. If this is a request message and none of the above are true, then the message body length is zero (no message body is present).

6. これが要求メッセージであり、上記のいずれにも該当しない場合、メッセージ本文の長さはゼロです（メッセージ本文は存在しません）。

7. Otherwise, this is a response message without a declared message body length, so the message body length is determined by the number of octets received prior to the server closing the connection.

7. それ以外の場合、これはメッセージ本文の長さが宣言されていない応答メッセージであるため、メッセージ本文の長さは、サーバーが接続を閉じる前に受信したオクテットの数によって決まります。

Since there is no way to distinguish a successfully completed, close-delimited message from a partially received message interrupted by network failure, a server SHOULD generate encoding or length-delimited messages whenever possible. The close-delimiting feature exists primarily for backwards compatibility with HTTP/1.0.

正常に完了したクローズ区切りメッセージを、ネットワーク障害によって中断された部分的に受信されたメッセージと区別する方法がないため、サーバーは、可能な場合は常にエンコードメッセージまたは長さ区切りメッセージを生成する必要があります。クローズ区切り機能は、主にHTTP / 1.0との下位互換性のために存在します。

A server MAY reject a request that contains a message body but not a Content-Length by responding with 411 (Length Required).

サーバーは、411（Length Required）で応答することにより、メッセージ本文を含むがContent-Lengthを含まないリクエストを拒否してもよい（MAY）。

Unless a transfer coding other than chunked has been applied, a client that sends a request containing a message body SHOULD use a valid Content-Length header field if the

チャンク以外の転送コーディングが適用されていない限り、メッセージ本文を含むリクエストを送信するクライアントは、チャンク転送コーディングではなく、メッセージ本文の長さが事

message body length is known in advance, rather than the chunked transfer coding, since some existing services respond to chunked with a 411 (Length Required) status code even though they understand the chunked transfer coding. This is typically because such services are implemented via a gateway that requires a content-length in advance of being called and the server is unable or unwilling to buffer the entire request before processing.

前にわかっている場合は、有効なContent-Lengthヘッダーフィールドを使用する必要があります。サービスは、チャンク転送コーディングを理解していても、411（長さを要求）ステータスコードでチャンク応答します。これは通常、そのようなサービスが呼び出される前にcontent-lengthを必要とするゲートウェイを介して実装され、サーバーが処理前にリクエスト全体をバッファリングできないか、またはしたくないためです。

A user agent that sends a request containing a message body MUST send a valid Content-Length header field if it does not know the server will handle HTTP/1.1 (or later) requests; such knowledge can be in the form of specific user configuration or by remembering the version of a prior received response.

メッセージ本文を含むリクエストを送信するユーザーエージェントは、サーバーがHTTP / 1.1（またはそれ以降）のリクエストを処理することを知らない場合、有効なContent-Lengthヘッダーフィールドを送信する必要があります。このような知識は、特定のユーザー構成の形式にすることも、以前に受け取った応答のバージョンを記憶することもできます。

If the final response to the last request on a connection has been completely received and there remains additional data to read, a user agent MAY discard the remaining data or attempt to determine if that data belongs as part of the prior response body, which might be the case if the prior message's Content-Length value is incorrect. A client MUST NOT process, cache, or forward such extra data as a separate response, since such behavior would be vulnerable to cache poisoning.

接続の最後のリクエストに対する最終応答が完全に受信され、読み取る追加データが残っている場合、ユーザーエージェントは残りのデータを破棄するか、そのデータが前の応答本文の一部として属しているかどうかを判断しようとします。前のメッセージのContent-Length値が正しくない場合。そのような動作はキャッシュポイズニングに対して脆弱であるため、クライアントはこのような追加データを個別の応答として処理、キャッシュ、または転送してはなりません（MUST NOT）。

3.4. Handling Incomplete Messages

3.4. 不完全なメッセージの処理

A server that receives an incomplete request message, usually due to a canceled request or a triggered timeout exception, MAY send an error response prior to closing the connection.

通常、キャンセルされた要求またはトリガーされたタイムアウト例外が原因で、不完全な要求メッセージを受信するサーバーは、接続を閉じる前にエラー応答を送信できます（MAY）。

A client that receives an incomplete response message, which can occur when a connection is closed prematurely or when decoding a supposedly chunked transfer coding fails, MUST record the message as incomplete. Cache requirements for incomplete responses are defined in Section 3 of [RFC7234].

不完全な応答メッセージを受信するクライアントは、接続が途中で閉じられた場合、またはチャンクされたと思われる転送コーディングのデコードが失敗した場合に発生する可能性があり、メッセージを不完全として記録する必要があります。不完全な応答のキャッシュ要件は、[RFC7234]のセクション3で定義されています。

If a response terminates in the middle of the header section (before the empty line is received) and the status code might rely on header fields to convey the full meaning of the response, then the client cannot assume that meaning has been conveyed; the client might need to repeat the request in order to determine what action to take next.

ヘッダーセクションの途中で（空の行が受信される前に）応答が終了し、ステータスコードがヘッダーフィールドに依存して応答の完全な意味を伝える場合、クライアントはその意味が伝えられたと想定できません。クライアントは、次に実行するアクションを決定するために、要求を繰り返す必要がある場合があります。

A message body that uses the chunked transfer coding is incomplete if the zero-sized chunk that terminates the encoding has not been received. A message that uses a valid Content-Length is incomplete if the size of the message body received (in octets) is less than the value given by Content-Length. A response that has neither chunked transfer coding nor Content-Length is terminated by closure of the connection and, thus, is considered complete regardless of the number of message body octets received, provided that the header section was received intact.

エンコードを終了するサイズがゼロのチャンクが受信されていない場合、チャンク転送コーディングを使用するメッセージ本文は不完全です。受信したメッセージ本文のサイズ（オクテット単位）がContent-Lengthで指定された値より小さい場合、有効なContent-Lengthを使用するメッセージは不完全です。チャンク転送コーディングもContent-Lengthもない応答は、接続のクローズによって終了するため、受信されたメッセージ本文のオクテットの数に関係なく、ヘッダーセクションがそのまま受信された場合、完了したと見なされます。

3.5. Message Parsing Robustness

3.5. メッセージ解析の堅牢性

Older HTTP/1.0 user agent implementations might send an extra CRLF after a POST request as a workaround for some early server applications that failed to read message body content that was not terminated by a line-ending. An HTTP/1.1 user agent MUST NOT preface or follow a request with an extra CRLF. If terminating the request message body with a line-ending is desired, then the user agent MUST count the terminating CRLF octets as part of the message body length.

古いHTTP / 1.0ユーザーエージェントの実装では、行末で終了していないメッセージ本文のコンテンツを読み取れなかった一部の初期のサーバーアプリケーションの回避策として、POST要求の後に追加のCRLFを送信する場合があります。 HTTP / 1.1ユーザーエージェントは、追加のCRLFを使用してリクエストの前置または後続してはなりません（MUST NOT）。要求メッセージ

本文を行末で終了することが望まれる場合、ユーザーエージェントは、終了CRLFオクテットをメッセージ本文の長さの一部としてカウントしなければなりません（MUST）。

In the interest of robustness, a server that is expecting to receive and parse a request-line SHOULD ignore at least one empty line (CRLF) received prior to the request-line.

堅牢性のために、要求行を受信して解析することを期待しているサーバーは、要求行の前に受信された少なくとも1つの空行（CRLF）を無視する必要があります（SHOULD）。

Although the line terminator for the start-line and header fields is the sequence CRLF, a recipient MAY recognize a single LF as a line terminator and ignore any preceding CR.

start-lineおよびheaderフィールドの行ターミネーターはシーケンスCRLFですが、受信者は単一のLFを行ターミネーターとして認識し、先行するCRを無視してもよい（MAY）。

Although the request-line and status-line grammar rules require that each of the component elements be separated by a single SP octet, recipients MAY instead parse on whitespace-delimited word boundaries and, aside from the CRLF terminator, treat any form of whitespace as the SP separator while ignoring preceding or trailing whitespace; such whitespace includes one or more of the following octets: SP, HTAB, VT (%x0B), FF (%x0C), or bare CR. However, lenient parsing can result in security vulnerabilities if there are multiple recipients of the message and each has its own unique interpretation of robustness (see Section 9.5).

request-lineとstatus-lineの文法規則では、各コンポーネント要素を単一のSPオクテットで区切る必要がありますが、受信者は代わりに空白で区切られた単語の境界で解析し、CRLFターミネーターを除いて、あらゆる形式の空白を先行または後続の空白を無視するときのSPセパレータ。このような空白には、SP、HTAB、VT（%x0B）、FF（%x0C）、またはベアCRの1つ以上のオクテットが含まれます。ただし、緩やかな解析では、メッセージの受信者が複数あり、それぞれに独自の堅牢性の解釈がある場合（セクション9.5を参照）、セキュリティが脆弱になる可能性があります。

When a server listening only for HTTP request messages, or processing what appears from the start-line to be an HTTP request message, receives a sequence of octets that does not match the HTTP-message grammar aside from the robustness exceptions listed above, the server SHOULD respond with a 400 (Bad Request) response.

サーバーがHTTPリクエストメッセージのみをリッスンしている場合、または開始行からHTTPリクエストメッセージであると思われるものを処理している場合、上記の堅牢性の例外を除いて、HTTPメッセージの文法と一致しないオクテットのシーケンスを受信します。 400（Bad Request）応答で応答する必要があります（SHOULD）。

4. Transfer Codings

4. 転送コーディング

Transfer coding names are used to indicate an encoding transformation that has been, can be, or might need to be applied to a payload body in order to ensure "safe transport" through the network. This differs from a content coding in that the transfer coding is a property of the message rather than a property of the representation that is being transferred.

転送コーディング名は、ネットワークを介した「安全なトランスポート」を保証するために、ペイロード本体に適用されている、適用できる、または適用する必要がある可能性があるエンコーディング変換を示すために使用されます。これは、転送コーディングが転送される表現のプロパティではなくメッセージのプロパティであるという点で、コンテンツコーディングとは異なります。

```
    transfer-coding    = "chunked" ; Section 4.1
                       / "compress" ; Section 4.2.1
                       / "deflate" ; Section 4.2.2
                       / "gzip" ; Section 4.2.3
                       / transfer-extension
    transfer-extension = token *( OWS ";" OWS transfer-parameter )
```

Parameters are in the form of a name or name=value pair.

パラメータは、名前または名前=値のペアの形式です。

```
    transfer-parameter = token BWS "=" BWS ( token / quoted-string )
```

All transfer-coding names are case-insensitive and ought to be registered within the HTTP Transfer Coding registry, as defined in Section 8.4. They are used in the TE (Section 4.3) and Transfer-Encoding (Section 3.3.1) header fields.

すべての転送コーディング名は大文字と小文字を区別せず、セクション8.4で定義されているように、HTTP転送コーディングレジストリ内に登録する必要があります。これらは、TE（セクション4.3）およびTransfer-Encoding（セクション3.3.1）ヘッダーフィールドで使用されます。

4.1. Chunked Transfer Coding

4.1. チャンク転送コーディング

The chunked transfer coding wraps the payload body in order to transfer it as a series of chunks, each with its own size indicator, followed by an OPTIONAL trailer containing header fields. Chunked enables content streams of unknown size to be transferred as a sequence of length-delimited buffers, which enables the sender to retain connection persistence and the recipient to know when it has received the entire message.

チャンク転送コーディングは、ペイロードボディをラップして一連のチャンクとして転送します。チャンクにはそれぞれ独自のサイズインジケータがあり、その後にヘッダーフィールドを含むオプションのトレーラーが続きます。 Chunkedを使用すると、不明なサイズのコンテンツストリームを長さ区切りの一連

のバッファーとして転送できます。これにより、送信者は接続の永続性を保持し、受信者はメッセージ全体をいつ受信したかを知ることができます。

```
    chunked-body   = *chunk
                     last-chunk
                     trailer-part
                     CRLF
```

chunk = chunk-size [chunk-ext] CRLF chunk-data CRLF chunk-size = 1*HEXDIG last-chunk = 1*("0") [chunk-ext] CRLF

chunk = chunk-size [chunk-ext] CRLF chunk-data CRLF chunk-size = 1 * HEXDIG last-chunk = 1 *（ "0"）[chunk-ext] CRLF

```
    chunk-data     = 1*OCTET ; a sequence of chunk-size octets
```

The chunk-size field is a string of hex digits indicating the size of the chunk-data in octets. The chunked transfer coding is complete when a chunk with a chunk-size of zero is received, possibly followed by a trailer, and finally terminated by an empty line.

チャンクサイズフィールドは、チャンクデータのサイズをオクテットで示す16進数の文字列です。チャンクサイズがゼロのチャンクが受信されたときにチャンク転送コーディングが完了し、トレーラーが続く場合があり、最後に空の行で終了します。

A recipient MUST be able to parse and decode the chunked transfer coding.

受信者は、チャンク転送コーディングを解析およびデコードできる必要があります。

4.1.1. Chunk Extensions

4.1.1. チャンク拡張

The chunked encoding allows each chunk to include zero or more chunk extensions, immediately following the chunk-size, for the sake of supplying per-chunk metadata (such as a signature or hash), mid-message control information, or randomization of message body size.

チャンクエンコーディングでは、チャンクごとのメタデータ（署名やハッシュなど）、メッセージの中間制御情報、またはメッセージ本文のランダム化を提供するために、各チャンクにチャンクサイズの直後に0個以上のチャンク拡張を含めることができますサイズ。

```
    chunk-ext      = *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
```

chunk-ext-name = token chunk-ext-val = token / quoted-string

chunk-ext-name =トークンchunk-ext-val =トークン/引用文字列

The chunked encoding is specific to each connection and is likely to be removed or recoded by each recipient (including intermediaries) before any higher-level application would have a chance to inspect the extensions. Hence, use of chunk extensions is generally limited to specialized HTTP services such as "long polling" (where client and server can have shared expectations regarding the use of chunk extensions) or for padding within an end-to-end secured connection.

チャンクエンコーディングは各接続に固有であり、上位レベルのアプリケーションが拡張機能を検査する前に、各受信者（仲介者を含む）によって削除または再コード化される可能性があります。したがって、チャンク拡張の使用は通常、「ロングポーリング」（クライアントとサーバーがチャンク拡張の使用に関して期待を共有できる）などの特殊なHTTPサービス、またはエンドツーエンドの安全な接続内のパディングに限定されます。

A recipient MUST ignore unrecognized chunk extensions. A server ought to limit the total length of chunk extensions received in a request to an amount reasonable for the services provided, in the same way that it applies length limitations and timeouts for other parts of a message, and generate an appropriate 4xx (Client Error) response if that amount is exceeded.

受信者は、認識されていないチャンク拡張を無視する必要があります。サーバーは、リクエストで受信したチャンク拡張の全長を、メッセージの他の部分に長さ制限とタイムアウトを適用するのと同じ方法で、提供されるサービスに適した量に制限し、適切な4xx（Client Error ）その量を超えた場合の応答。

4.1.2. Chunked Trailer Part

4.1.2. チャンクトレーラーパーツ

A trailer allows the sender to include additional fields at the end of a chunked message in order to supply metadata that might be dynamically generated while the message body is sent, such as a message integrity check, digital signature, or post-processing status. The trailer fields are identical to header fields, except they are sent in a chunked trailer instead of the message's header section.

トレーラーを使用すると、メッセージの完全性チェック、デジタル署名、後処理ステータスなど、メッセージ本文の送信中に動的に生成される可能性のあるメタデータを提供するために、チャンクされたメッセージの末尾に追加フィールドを含めることができます。トレーラーフィールドは、メッセージのヘッダーセクションではなくチャンクトレーラーで送信されることを除いて、ヘッダーフィールドと同じです。

```
    trailer-part   = *( header-field CRLF )
```

A sender MUST NOT generate a trailer that contains a field necessary for message framing (e.g., Transfer-Encoding and Content-Length), routing (e.g., Host), request modifiers (e.g., controls and conditionals in Section 5 of [RFC7231]), authentication (e.g., see [RFC7235] and [RFC6265]), response control data (e.g., see Section 7.1 of [RFC7231]), or determining how to process the payload (e.g., Content-Encoding, Content-Type, Content-Range, and Trailer).

送信者は、メッセージのフレーミング（たとえば、Transfer-EncodingおよびContent-Length）、ルーティング（たとえば、ホスト）、要求修飾子（たとえば、[RFC7231]のセクション5のコントロールおよび条件）に必要なフィールドを含むトレーラーを生成してはなりません（MUST NOT）。、認証（例：[RFC7235]および[RFC6265]を参照）、応答制御データ（例：[RFC7231]のセクション7.1を参照）、またはペイロードの処理方法（例：Content-Encoding、Content-Type、Content-範囲、およびトレーラー）。

When a chunked message containing a non-empty trailer is received, the recipient MAY process the fields (aside from those forbidden above) as if they were appended to the message's header section. A recipient MUST ignore (or consider as an error) any fields that are forbidden to be sent in a trailer, since processing them as if they were present in the header section might bypass external security filters.

空でないトレーラーを含むチャンクメッセージが受信されると、受信者はフィールド（上記で禁止されているものを除く）を、メッセージのヘッダーセクションに追加された場合と同様に処理できます（MAY）。ヘッダーセクションに存在するかのようにフィールドを処理すると外部セキュリティフィルターがバイパスされる可能性があるため、受信者はトレーラーでの送信が禁止されているフィールドを無視する（またはエラーと見なす）必要があります。

Unless the request includes a TE header field indicating "trailers" is acceptable, as described in Section 4.3, a server SHOULD NOT generate trailer fields that it believes are necessary for the user agent to receive. Without a TE containing "trailers", the server ought to assume that the trailer fields might be silently discarded along the path to the user agent. This requirement allows intermediaries to forward a de-chunked message to an HTTP/1.0 recipient without buffering the entire response.

セクション4.3で説明されているように、リクエストに「予告編」が受け入れられることを示すTEヘッダーフィールドが含まれていない限り、サーバーは、ユーザーエージェントが受信する必要があると考える予告編フィールドを生成してはなりません（SHOULD NOT）。「トレーラー」を含むTEがない場合、サーバーは、トレーラーフィールドがユーザーエージェントへのパスに沿って静かに破棄される可能性があると想定する必要があります。この要件により、仲介者は、チャンク化されたメッセージを応答全体をバッファリングせずにHTTP / 1.0受信者に転送できます。

4.1.3. Decoding Chunked

A process for decoding the chunked transfer coding can be represented in pseudo-code as:

4.1.3. チャンクのデコード

チャンク転送コーディングをデコードするプロセスは、擬似コードで次のように表すことができます。

```
    length := 0
    read chunk-size, chunk-ext (if any), and CRLF
    while (chunk-size > 0) {
       read chunk-data and CRLF
       append chunk-data to decoded-body
       length := length + chunk-size
       read chunk-size, chunk-ext (if any), and CRLF
    }
    read trailer field
    while (trailer field is not empty) {
       if (trailer field is allowed to be sent in a trailer) {
           append trailer field to existing header fields
       }
       read trailer-field
    }
    Content-Length := length
    Remove "chunked" from Transfer-Encoding
    Remove Trailer from existing header fields
```

4.2. Compression Codings

The codings defined below can be used to compress the payload of a message.

4.2. 圧縮コーディング

以下に定義するコーディングは、メッセージのペイロードを圧縮するために使用できます。

4.2.1. Compress Coding

The "compress" coding is an adaptive Lempel-Ziv-Welch (LZW) coding [Welch] that is commonly produced by the UNIX file compression program "compress". A recipient SHOULD consider "x-compress" to be equivalent to "compress".

4.2.1. 圧縮コーディング

「圧縮」コーディングは、UNIXファイル圧縮プログラム「圧縮」によって一般的に生成される適応型レンペル-ジブ-ウェルチ（LZW）コーディング[ウェルチ]です。受信者は、「x-compress」を「compress」と同等であると見なすべきです。

4.2.2. Deflate Coding

The "deflate" coding is a "zlib" data format [RFC1950] containing a "deflate" compressed data stream [RFC1951] that

4.2.2. デフレートコーディング

「deflate」コーディングは、Lempel-Ziv（LZ77）圧縮アルゴリズムとハフマンコーディングの組み合わせを使用する

uses a combination of the Lempel-Ziv (LZ77) compression algorithm and Huffman coding.

Note: Some non-conformant implementations send the "deflate" compressed data without the zlib wrapper.

「deflate」圧縮データストリーム[RFC1951]を含む「zlib」データフォーマット[RFC1950]です。

注：一部の非準拠の実装は、zlibラッパーなしで「deflate」圧縮データを送信します。

4.2.3. Gzip Coding

The "gzip" coding is an LZ77 coding with a 32-bit Cyclic Redundancy Check (CRC) that is commonly produced by the gzip file compression program [RFC1952]. A recipient SHOULD consider "x-gzip" to be equivalent to "gzip".

4.2.3. Gzipコーディング

"gzip"コーディングは、gzipファイル圧縮プログラム[RFC1952]によって一般的に生成される32ビット巡回冗長検査（CRC）を備えたLZ77コーディングです。受信者は、「x-gzip」を「gzip」と同等であると見なすべきです。

4.3. TE

The "TE" header field in a request indicates what transfer codings, besides chunked, the client is willing to accept in response, and whether or not the client is willing to accept trailer fields in a chunked transfer coding.

The TE field-value consists of a comma-separated list of transfer coding names, each allowing for optional parameters (as described in Section 4), and/or the keyword "trailers". A client MUST NOT send the chunked transfer coding name in TE; chunked is always acceptable for HTTP/1.1 recipients.

4.3. 手

リクエストの「TE」ヘッダーフィールドは、チャンク以外の転送コーディング、クライアントが応答として受け入れる用意があるか、およびクライアントがチャンク転送コーディングのトレーラーフィールドを受け入れるかどうかを示します。

TEフィールド値は、転送コーディング名のコンマ区切りのリストで構成され、それぞれがオプションのパラメーター（セクション4で説明）やキーワード「予告編」を許可します。クライアントは、チャンク化された転送コーディング名をTEで送信してはなりません（MUST NOT）。チャンクは、HTTP / 1.1受信者には常に受け入れられます。

```
    TE        = #t-codings
    t-codings = "trailers" / ( transfer-coding [ t-ranking ] )
    t-ranking = OWS ";" OWS "q=" rank
    rank      = ( "0" [ "." 0*3DIGIT ] )
              / ( "1" [ "." 0*3("0") ] )
```

Three examples of TE use are below.

THEの3つの例を以下に示します。

```
    TE: deflate
    TE:
    TE: trailers, deflate;q=0.5
```

The presence of the keyword "trailers" indicates that the client is willing to accept trailer fields in a chunked transfer coding, as defined in Section 4.1.2, on behalf of itself and any downstream clients. For requests from an intermediary, this implies that either: (a) all downstream clients are willing to accept trailer fields in the forwarded response; or, (b) the intermediary will attempt to buffer the response on behalf of downstream recipients. Note that HTTP/1.1 does not define any means to limit the size of a chunked response such that an intermediary can be assured of buffering the entire response.

キーワード「予告編」の存在は、クライアントが、セクション4.1.2で定義されているように、自分自身とダウンストリームクライアントに代わってチャンク転送コーディングで予告フィールドを受け入れる用意があることを示します。仲介者からのリクエストの場合、これは次のいずれかを意味します。（a）すべてのダウンストリームクライアントが転送された応答のトレーラーフィールドを受け入れる用意がある。または、（b）仲介者がダウンストリームの受信者に代わって応答をバッファしようとします。 HTTP / 1.1は、中間体が応答全体を確実にバッファリングできるように、チャンク化された応答のサイズを制限する手段を定義していないことに注意してください。

When multiple transfer codings are acceptable, the client MAY rank the codings by preference using a case-insensitive "q" parameter (similar to the qvalues used in content negotiation fields, Section 5.3.1 of [RFC7231]). The rank value is a real number in the range 0 through 1, where 0.001 is the least preferred and 1 is the most preferred; a value of 0 means "not acceptable".

複数の転送コーディングが受け入れられる場合、クライアントは、大文字と小文字を区別しない「q」パラメーターを使用して、コーディングを優先的にランク付けできます（コンテンツネゴシエーションフィールドで使用されるqvalueと同様、[RFC7231]のセクション5.3.1）。ランク値は0から1の範囲の実数で、0.001が最も優先されず、1が最も優先されます。値0は「受け入れられない」ことを意味します。

If the TE field-value is empty or if no TE field is present, the only acceptable transfer coding is chunked. A message with no transfer coding is always acceptable.

TEフィールド値が空の場合、またはTEフィールドが存在しない場合、受け入れ可能な転送コーディングのみがチャンク化されます。転送コーディングのないメッセージは常に受け入れられます。

Since the TE header field only applies to the immediate connection, a sender of TE MUST also send a "TE" connection option within the Connection header field (Section 6.1) in order

TEヘッダーフィールドは即時接続にのみ適用されるため、TEの送信者は、TEフィールドがサポートしていない仲介者によって転送されないようにするために、接続ヘッダーフィールド（セ

to prevent the TE field from being forwarded by intermediaries that do not support its semantics.

クション6.1）内で「TE」接続オプションも送信する必要があります。そのセマンティクス。

4.4. Trailer

When a message includes a message body encoded with the chunked transfer coding and the sender desires to send metadata in the form of trailer fields at the end of the message, the sender SHOULD generate a Trailer header field before the message body to indicate which fields will be present in the trailers. This allows the recipient to prepare for receipt of that metadata before it starts processing the body, which is useful if the message is being streamed and the recipient wishes to confirm an integrity check on the fly.

Trailer = 1#field-name

4.4. トレーラー

メッセージにチャンク転送コーディングでエンコードされたメッセージ本文が含まれていて、送信者がメッセージの末尾にあるトレーラーフィールドの形式でメタデータを送信したい場合、送信者は、メッセージ本文の前にトレーラーヘッダーフィールドを生成して、どのフィールドがトレーラーにいること。これにより、受信者は本文の処理を開始する前にそのメタデータの受信を準備できます。これは、メッセージがストリーミングされており、受信者がオンザフライで整合性チェックを確認したい場合に役立ちます。

予告編= 1＃フィールド名

5. Message Routing

HTTP request message routing is determined by each client based on the target resource, the client's proxy configuration, and establishment or reuse of an inbound connection. The corresponding response routing follows the same connection chain back to the client.

5. メッセージルーティング

HTTP要求メッセージのルーティングは、ターゲットリソース、クライアントのプロキシ構成、およびインバウンド接続の確立または再利用に基づいて、各クライアントによって決定されます。対応する応答ルーティングは、同じ接続チェーンに従ってクライアントに戻ります。

5.1. Identifying a Target Resource

HTTP is used in a wide variety of applications, ranging from general-purpose computers to home appliances. In some cases, communication options are hard-coded in a client's configuration. However, most HTTP clients rely on the same resource identification mechanism and configuration techniques as general-purpose Web browsers.

HTTP communication is initiated by a user agent for some purpose. The purpose is a combination of request semantics, which are defined in [RFC7231], and a target resource upon which to apply those semantics. A URI reference (Section 2.7) is typically used as an

identifier for the "target resource", which a user agent would resolve to its absolute form in order to obtain the "target URI". The target URI excludes the reference's fragment component, if any, since fragment identifiers are reserved for client-side processing ([RFC3986], Section 3.5).

5.1. ターゲットリソースの特定

HTTPは、汎用コンピュータから家電製品に至るまで、さまざまなアプリケーションで使用されています。場合によっては、通信オプションはクライアントの構成にハードコーディングされます。ただし、ほとんどのHTTPクライアントは、汎用のWebブラウザーと同じリソース識別メカニズムと構成手法に依存しています。

HTTP通信は、何らかの目的でユーザーエージェントによって開始されます。その目的は、[RFC7231]で定義されている要求のセマンティクスと、それらのセマンティクスを適用するターゲットリソースの組み合わせです。 URI参照（セクション2.7）は通常、

ユーザーエージェントが「ターゲットURI」を取得するためにその絶対形式に解決する「ターゲットリソース」の識別子。フラグメント識別子はクライアント側の処理用に予約されているため（[RFC3986]、セクション3.5）、ターゲットURIは参照のフラグメントコンポーネントを除外します（ある場合）。

5.2. Connecting Inbound

Once the target URI is determined, a client needs to decide whether a network request is necessary to accomplish the desired semantics and, if so, where that request is to be directed.

If the client has a cache [RFC7234] and the request can be satisfied by it, then the request is usually directed there first.

If the request is not satisfied by a cache, then a typical client will check its configuration to determine whether a proxy is to be used to satisfy the request. Proxy configuration is implementation-dependent, but is often based on URI prefix matching, selective authority matching, or both, and the proxy itself is usually identified by an "http" or "https" URI. If a proxy is applicable, the client connects inbound by establishing (or reusing) a connection to that proxy.

If no proxy is applicable, a typical client will invoke a handler

5.2. インバウンドの接続

ターゲットURIが決定されると、クライアントは、目的のセマンティクスを実現するためにネットワーク要求が必要かどうかを決定する必要があり、必要な場合は、その要求の送信先を決定する必要があります。

クライアントにキャッシュ[RFC7234]があり、それによって要求を満たすことができる場合、要求は通常、最初にそこに送られます。

要求がキャッシュで満たされない場合、一般的なクライアントはその構成をチェックして、要求を満たすためにプロキシを使用するかどうかを決定します。プロキシ設定は実装に依存しますが、多くの場合、URIプレフィックスの一致、選択的な権限の一致、またはその両方に基づいており、プロキシ自体は通常「http」または「https」URIで識別されます。プロキシが適用可能な場合、クライアントはそのプロキシへの接続を確立（または再利用）することにより、インバウンドに接続します。

プロキシが適用できない場合、通常のクライアントは、通常は

routine, usually specific to the target URI's scheme, to connect directly to an authority for the target resource. How that is accomplished is dependent on the target URI scheme and defined by its associated specification, similar to how this specification defines origin server access for resolution of the "http" (Section 2.7.1) and "https" (Section 2.7.2) schemes.

ターゲットURIのスキームに固有のハンドラルーチンを呼び出して、ターゲットリソースの権限に直接接続します。これがどのように達成されるかは、ターゲットURIスキームに依存し、関連する仕様によって定義されます。これは、この仕様が「http」（セクション2.7.1）および「https」（セクション2.7.2）の解決のためのオリジンサーバーアクセスを定義する方法と同様です。スキーム。

HTTP requirements regarding connection management are defined in Section 6.

接続管理に関するHTTP要件は、セクション6で定義されています。

5.3. Request Target

5.3. リクエスト対象

Once an inbound connection is obtained, the client sends an HTTP request message (Section 3) with a request-target derived from the target URI. There are four distinct formats for the request-target, depending on both the method being requested and whether the request is to a proxy.

インバウンド接続が取得されると、クライアントは、ターゲットURIから派生したリクエストターゲットを含むHTTPリクエストメッセージ（セクション3）を送信します。リクエストされるメソッドと、リクエストがプロキシに対するものかどうかに応じて、リクエストターゲットには4つの異なる形式があります。

```
    request-target = origin-form
                   / absolute-form
                   / authority-form
                   / asterisk-form
```

5.3.1. origin-form

5.3.1. 原点フォーム

The most common form of request-target is the origin-form.

request-targetの最も一般的な形式はorigin-formです。

origin-form = absolute-path ["?" query]

origin-form =絶対パス[" ？ "クエリ]

When making a request directly to an origin server, other than a CONNECT or server-wide OPTIONS request (as detailed below), a client MUST send only the absolute path and query components of the target URI as the request-target. If the target URI's path component is empty, the client MUST send "/" as the path within the origin-form of request-target. A Host header field is also sent, as defined in Section 5.4.

CONNECTまたはサーバー全体のOPTIONSリクエスト（以下で詳述）以外で、オリジンサーバーに直接リクエストを行う場合、クライアントは絶対パスとターゲットURIのクエリコンポーネントのみをリクエストターゲットとして送信する必要があります。ターゲットURIのパスコンポーネントが空の場合、クライアントは、リクエストターゲットのorigin-form内のパスとして「/」を送信する必要があります。セクション5.4で定義されているように、ホストヘッダーフィールドも送信されます。

For example, a client wishing to retrieve a representation of the resource identified as

たとえば、次のように識別されたリソースの表現を取得したいクライアント

```
    http://www.example.org/where?q=now
```

directly from the origin server would open (or reuse) a TCP connection to port 80 of the host "www.example.org" and send the lines:

オリジンサーバーから直接、ホスト「www.example.org」のポート80へのTCP接続を開き（または再利用）、次の行を送信します。

```
    GET /where?q=now HTTP/1.1
    Host: www.example.org
```

followed by the remainder of the request message.

残りの要求メッセージが続きます。

5.3.2. absolute-form

5.3.2. 絶対形

When making a request to a proxy, other than a CONNECT or server-wide OPTIONS request (as detailed below), a client MUST send the target URI in absolute-form as the request-target.

CONNECTまたはサーバー全体のOPTIONSリクエスト以外のプロキシへのリクエストを行う場合（詳細は以下を参照）、クライアントは絶対ターゲット形式のターゲットURIをリクエストターゲットとして送信する必要があります。

absolute-form = absolute-URI

絶対形式=絶対URI

The proxy is requested to either service that request from a valid cache, if possible, or make the same request on the client's behalf to either the next inbound proxy server or

プロキシは、可能であれば、有効なキャッシュからの要求にサービスを提供するか、クライアントに代わって次のインバウンドプロキシサーバーに、または要求ターゲットによって示され

directly to the origin server indicated by the request-target. Requirements on such "forwarding" of messages are defined in Section 5.7.

An example absolute-form of request-line would be:

```
GET http://www.example.org/pub/WWW/TheProject.html HTTP/1.1
```

To allow for transition to the absolute-form for all requests in some future version of HTTP, a server MUST accept the absolute-form in requests, even though HTTP/1.1 clients will only send them in requests to proxies.

5.3.3. authority-form

The authority-form of request-target is only used for CONNECT requests (Section 4.3.6 of [RFC7231]).

authority-form = authority

When making a CONNECT request to establish a tunnel through one or more proxies, a client MUST send only the target URI's authority component (excluding any userinfo and its "@" delimiter) as the request-target. For example,

CONNECT www.example.com:80 HTTP/1.1

5.3.4. asterisk-form

The asterisk-form of request-target is only used for a server-wide OPTIONS request (Section 4.3.7 of [RFC7231]).

asterisk-form = "*"

When a client wishes to request OPTIONS for the server as a whole, as opposed to a specific named resource of that server, the client MUST send only "*" (%x2A) as the request-target. For example,

OPTIONS * HTTP/1.1

If a proxy receives an OPTIONS request with an absolute-form of request-target in which the URI has an empty path and no query component, then the last proxy on the request chain MUST send a request-target of "*" when it forwards the request to the indicated origin server.

For example, the request

```
OPTIONS http://www.example.org:8001 HTTP/1.1
```

would be forwarded by the final proxy as

```
OPTIONS * HTTP/1.1
Host: www.example.org:8001
```

after connecting to port 8001 of host "www.example.org".

5.4. Host

The "Host" header field in a request provides the host and port information from the target URI, enabling the origin server to distinguish among resources while servicing requests for multiple host names on a single IP address.

```
Host = uri-host [ ":" port ] ; Section 2.7.1
```

るオリジンサーバーに直接、同じ要求を行うように要求されます。このようなメッセージの「転送」に関する要件は、セクション5.7で定義されています。

request-lineの絶対形式の例は次のようになります。

将来のバージョンのHTTPですべての要求を絶対形式に移行できるようにするには、HTTP / 1.1クライアントがプロキシへの要求でのみ送信する場合でも、サーバーは要求で絶対形式を受け入れる必要があります。

5.3.3. 権限フォーム

権限形式のリクエストターゲットは、CONNECTリクエストにのみ使用されます（[RFC7231]のセクション4.3.6）。

権限フォーム=権限

1つまたは複数のプロキシを介してトンネルを確立するCONNECT要求を行う場合、クライアントは、ターゲットURIの権限コンポーネント（userinfoおよびその「@」区切り文字を除く）のみを要求ターゲットとして送信する必要があります。例えば、

接続www.example.com:80 HTTP / 1.1

5.3.4. アスタリスク形式

request-targetのアスタリスク形式は、サーバー全体のOPTIONSリクエストにのみ使用されます（[RFC7231]のセクション4.3.7）。

アスタリスク形式= "*"

クライアントがそのサーバーの特定の名前付きリソースではなく、サーバー全体のOPTIONSを要求する場合、クライアントは要求ターゲットとして "*"（%x2A）のみを送信する必要があります。例えば、

オプション* HTTP / 1.1

プロキシが絶対形式のリクエストターゲットのOPTIONSリクエストを受信した場合、URIには空のパスがあり、クエリコンポーネントはありません。リクエストチェーンの最後のプロキシは、転送時に「*」のリクエストターゲットを送信する必要があります。指定されたオリジンサーバーへのリクエスト。

たとえば、リクエスト

```
OPTIONS http://www.example.org:8001 HTTP/1.1
```

最終プロキシによって次のように転送されます

```
OPTIONS * HTTP/1.1
Host: www.example.org:8001
```

ホスト「www.example.org」のポート8001に接続した後。

5.4. ホスト

リクエストの「Host」ヘッダーフィールドは、ターゲットURIからのホストとポートの情報を提供し、オリジンサーバーが単一のIPアドレスで複数のホスト名のリクエストを処理しながらリソースを区別できるようにします。

A client MUST send a Host header field in all HTTP/1.1 request messages. If the target URI includes an authority component, then a client MUST send a field-value for Host that is identical to that authority component, excluding any userinfo subcomponent and its "@" delimiter (Section 2.7.1). If the authority component is missing or undefined for the target URI, then a client MUST send a Host header field with an empty field-value.

クライアントは、すべてのHTTP / 1.1要求メッセージでホストヘッダーフィールドを送信する必要があります。ターゲットURIにオーソリティコンポーネントが含まれている場合、クライアントは、userinfoサブコンポーネントとその「@」デリミタを除いて、そのオーソリティコンポーネントと同じHostのフィールド値を送信する必要があります（セクション2.7.1）。権限コンポーネントが見つからないか、ターゲットURIで定義されていない場合、クライアントは、空のフィールド値を持つホストヘッダーフィールドを送信する必要があります。

Since the Host field-value is critical information for handling a request, a user agent SHOULD generate Host as the first header field following the request-line.

Hostフィールド値はリクエストを処理するための重要な情報であるため、ユーザーエージェントは、リクエスト行に続く最初のヘッダーフィールドとしてホストを生成する必要があります（SHOULD）。

For example, a GET request to the origin server for <http://www.example.org/pub/WWW/> would begin with:

たとえば、<http://www.example.org/pub/WWW/>に対するオリジンサーバーへのGETリクエストは、次のように始まります。

```
GET /pub/WWW/ HTTP/1.1
Host: www.example.org
```

A client MUST send a Host header field in an HTTP/1.1 request even if the request-target is in the absolute-form, since this allows the Host information to be forwarded through ancient HTTP/1.0 proxies that might not have implemented Host.

リクエストターゲットが絶対形式であっても、クライアントはホストヘッダーフィールドをHTTP / 1.1リクエストで送信する必要があります。これにより、ホストを実装していない可能性のある古いHTTP / 1.0プロキシを通じてホスト情報を転送できるためです。

When a proxy receives a request with an absolute-form of request-target, the proxy MUST ignore the received Host header field (if any) and instead replace it with the host information of the request-target. A proxy that forwards such a request MUST generate a new Host field-value based on the received request-target rather than forward the received Host field-value.

プロキシが絶対形式の要求ターゲットを持つ要求を受信すると、プロキシは受信したホストヘッダーフィールド（存在する場合）を無視し、代わりにそれを要求ターゲットのホスト情報で置き換える必要があります。このようなリクエストを転送するプロキシは、受信したホストフィールド値を転送するのではなく、受信したリクエストターゲットに基づいて新しいホストフィールド値を生成する必要があります。

Since the Host header field acts as an application-level routing mechanism, it is a frequent target for malware seeking to poison a shared cache or redirect a request to an unintended server. An interception proxy is particularly vulnerable if it relies on the Host field-value for redirecting requests to internal servers, or for use as a cache key in a shared cache, without first verifying that the intercepted connection is targeting a valid IP address for that host.

Hostヘッダーフィールドはアプリケーションレベルのルーティングメカニズムとして機能するため、共有キャッシュを汚染したり、要求を意図しないサーバーにリダイレクトしたりしようとするマルウェアの頻繁なターゲットになります。インターセプトされた接続がそのホストの有効なIPアドレスをターゲットにしていることを最初に確認せずに、インターセプトプロキシが内部サーバーにリクエストをリダイレクトするため、または共有キャッシュのキャッシュキーとして使用するためにHostフィールド値に依存している場合、インターセプトプロキシは特に脆弱です。

A server MUST respond with a 400 (Bad Request) status code to any HTTP/1.1 request message that lacks a Host header field and to any request message that contains more than one Host header field or a Host header field with an invalid field-value.

サーバーは、ホストヘッダーフィールドがないHTTP / 1.1リクエストメッセージ、および複数のホストヘッダーフィールドまたは無効なフィールド値を持つホストヘッダーフィールドを含むリクエストメッセージに対して、400（不良リクエスト）ステータスコードで応答する必要があります。 。

5.5. Effective Request URI

5.5. 有効なリクエストURI

Since the request-target often contains only part of the user agent's target URI, a server reconstructs the intended target as an "effective request URI" to properly service the request. This reconstruction involves both the server's local configuration and information communicated in the request-target, Host header field, and connection context.

多くの場合、リクエストターゲットにはユーザーエージェントのターゲットURIの一部しか含まれていないため、サーバーは意図したターゲットを「有効なリクエストURI」として再構築して、リクエストを適切に処理します。この再構築には、サーバーのローカル構成と、リクエストターゲット、ホストヘッダーフィールド、および接続コンテキストで通信される情報の両方が含まれます。

For a user agent, the effective request URI is the target URI.

ユーザーエージェントの場合、有効なリクエストURIはターゲットURIです。

If the request-target is in absolute-form, the effective request

request-targetが絶対形式の場合、有効な要求URIはrequest-

URI is the same as the request-target. Otherwise, the effective request URI is constructed as follows:

targetと同じです。それ以外の場合、有効なリクエストURIは次のように構成されます。

If the server's configuration (or outbound gateway) provides a fixed URI scheme, that scheme is used for the effective request URI. Otherwise, if the request is received over a TLS-secured TCP connection, the effective request URI's scheme is "https"; if not, the scheme is "http".

サーバーの構成（または送信ゲートウェイ）が固定URIスキームを提供する場合、そのスキームが有効な要求URIに使用されます。それ以外の場合、要求がTLSで保護されたTCP接続を介して受信されると、有効な要求URIのスキームは「https」になります。そうでない場合、スキームは「http」です。

If the server's configuration (or outbound gateway) provides a fixed URI authority component, that authority is used for the effective request URI. If not, then if the request-target is in authority-form, the effective request URI's authority component is the same as the request-target. If not, then if a Host header field is supplied with a non-empty field-value, the authority component is the same as the Host field-value. Otherwise, the authority component is assigned the default name configured for the server and, if the connection's incoming TCP port number differs from the default port for the effective request URI's scheme, then a colon (":") and the incoming port number (in decimal form) are appended to the authority component.

サーバーの構成（または送信ゲートウェイ）が固定URI権限コンポーネントを提供する場合、その権限は有効な要求URIに使用されます。そうでない場合、要求ターゲットが権限形式の場合、有効な要求URIの権限コンポーネントは要求ターゲットと同じです。そうでない場合、ホストヘッダーフィールドに空でないフィールド値が指定されていると、機関コンポーネントはホストフィールド値と同じになります。それ以外の場合、認証コンポーネントにはサーバーに設定されたデフォルト名が割り当てられ、接続の受信TCPポート番号が有効なリクエストURIのスキームのデフォルトポートと異なる場合は、コロン（ "：") と受信ポート番号（ 10進形式）が機関コンポーネントに追加されます。

If the request-target is in authority-form or asterisk-form, the effective request URI's combined path and query component is empty. Otherwise, the combined path and query component is the same as the request-target.

request-targetがauthor-formまたはasterisk-formの場合、有効なリクエストURIのパスとクエリコンポーネントの組み合わせは空になります。それ以外の場合、結合されたパスとクエリコンポーネントは、リクエストターゲットと同じです。

The components of the effective request URI, once determined as above, can be combined into absolute-URI form by concatenating the scheme, "://", authority, and combined path and query component.

上記のように決定された有効な要求URIのコンポーネントは、スキーム、「：//」、権限、およびパスとクエリのコンポーネントを組み合わせることにより、絶対URI形式に組み合わせることができます。

Example 1: the following message received over an insecure TCP connection

例1：安全でないTCP接続で受信した次のメッセージ

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.example.org:8080
```

has an effective request URI of

の有効なリクエストURI

```
http://www.example.org:8080/pub/WWW/TheProject.html
```

Example 2: the following message received over a TLS-secured TCP connection

例2：TLSで保護されたTCP接続を介して受信した次のメッセージ

OPTIONS * HTTP/1.1 Host: www.example.org

オプション* HTTP / 1.1ホスト：www.example.org

has an effective request URI of

の有効なリクエストURI

```
https://www.example.org
```

Recipients of an HTTP/1.0 request that lacks a Host header field might need to use heuristics (e.g., examination of the URI path for something unique to a particular host) in order to guess the effective request URI's authority component.

Hostヘッダーフィールドのないhttp / 1.0リクエストの受信者は、効果的なリクエストURIのオーソリティコンポーネントを推測するために、ヒューリスティック（たとえば、特定のホストに固有のURIパスの調査）を使用する必要がある場合があります。

Once the effective request URI has been constructed, an origin server needs to decide whether or not to provide service for that URI via the connection in which the request was received. For example, the request might have been misdirected, deliberately or accidentally, such that the information within a received request-target or Host header field differs from the host or port upon which the connection has been made. If the connection is from a trusted gateway, that inconsistency might

有効なリクエストURIが構築されると、オリジンサーバーは、リクエストが受信された接続を介してそのURIにサービスを提供するかどうかを決定する必要があります。たとえば、受信した要求ターゲットまたはホストヘッダーフィールド内の情報が、接続が確立されたホストまたはポートと異なるように、要求が誤って、意図的にまたは誤って送信された可能性があります。接続が信頼できるゲートウェイからのものである場合、その不整合が予想される場合があります。それ以外の場合は、セキュリテ

be expected; otherwise, it might indicate an attempt to bypass security filters, trick the server into delivering non-public content, or poison a cache. See Section 9 for security considerations regarding message routing.

ィフィルターをバイパスする試み、サーバーを騙して非公開コンテンツを配信する試み、またはキャッシュを汚染する試みを示している可能性があります。メッセージルーティングに関するセキュリティの考慮事項については、セクション9を参照してください。

5.6. Associating a Response to a Request

HTTP does not include a request identifier for associating a given request message with its corresponding one or more response messages. Hence, it relies on the order of response arrival to correspond exactly to the order in which requests are made on the same connection. More than one response message per request only occurs when one or more informational responses (1xx, see Section 6.2 of [RFC7231]) precede a final response to the same request.

A client that has more than one outstanding request on a connection MUST maintain a list of outstanding requests in the order sent and MUST associate each received response message on that connection to the highest ordered request that has not yet received a final (non-1xx) response.

5.6. リクエストへのレスポンスの関連付け

HTTPには、特定の要求メッセージを対応する1つ以上の応答メッセージに関連付けるための要求識別子は含まれていません。したがって、同じ接続で要求が行われた順序に正確に対応するために、応答の到着順序に依存しています。リクエストごとに複数の応答メッセージが発生するのは、1つ以上の情報応答（1xx、[RFC7231]のセクション6.2を参照）が同じリクエストに対する最終応答の前にある場合のみです。

接続で複数の未処理の要求があるクライアントは、送信された順序で未処理の要求のリストを維持しなければならず、その接続で受信した各応答メッセージを、まだ最終（1xx以外）を受信していない最上位の要求に関連付けなければなりません（MUST）。応答。

5.7. Message Forwarding

As described in Section 2.3, intermediaries can serve a variety of roles in the processing of HTTP requests and responses. Some intermediaries are used to improve performance or availability. Others are used for access control or to filter content. Since an HTTP stream has characteristics similar to a pipe-and-filter architecture, there are no inherent limits to the extent an intermediary can enhance (or interfere) with either direction of the stream.

An intermediary not acting as a tunnel MUST implement the Connection header field, as specified in Section 6.1, and exclude fields from being forwarded that are only intended for the incoming connection.

An intermediary MUST NOT forward a message to itself unless it is protected from an infinite request loop. In general, an intermediary ought to recognize its own server names, including any aliases, local variations, or literal IP addresses, and respond to such requests directly.

5.7. メッセージ転送

2.3節で説明したように、仲介者はHTTP要求と応答の処理においてさまざまな役割を果たすことができます。一部の仲介者は、パフォーマンスまたは可用性を向上させるために使用されます。その他は、アクセス制御またはコンテンツのフィルタリングに使用されます。 HTTPストリームはパイプアンドフィルターアーキテクチャに似た特性を持っているため、仲介者がストリームのいずれかの方向を強化（または干渉）できる範囲に固有の制限はありません。

トンネルとして機能しない仲介者は、セクション6.1で指定されているように、接続ヘッダーフィールドを実装し、着信接続専用のフィールドを転送から除外する必要があります。

仲介者は、無限のリクエストループから保護されていない限り、メッセージを自分自身に転送してはなりません（MUST NOT）。一般に、仲介者は、エイリアス、ローカルバリエーション、またはリテラルIPアドレスを含む自身のサーバー名を認識し、そのような要求に直接応答する必要があります。

5.7.1. Via

The "Via" header field indicates the presence of intermediate protocols and recipients between the user agent and the server (on requests) or between the origin server and the client (on responses), similar to the "Received" header field in email (Section 3.6.7 of [RFC5322]). Via can be used for tracking message forwards, avoiding request loops, and identifying the protocol capabilities of senders along the request/response chain.

5.7.1. 経由

"Via"ヘッダーフィールドは、電子メールの "Received"ヘッダーフィールド（セクション[RFC5322]の3.6.7）。 viaは、メッセージ転送の追跡、要求ループの回避、および要求/応答チェーンに沿った送信者のプロトコル機能の識別に使用できます。

Via = 1#(received-protocol RWS received-by [RWS comment])

Via = 1＃（received-protocol RWS received-by [RWS comment]）

```
    received-protocol = [ protocol-name "/" ] protocol-version
                        ; see Section 6.7
    received-by       = ( uri-host [ ":" port ] ) / pseudonym
    pseudonym         = token
```

Multiple Via field values represent each proxy or gateway that has forwarded the message. Each intermediary appends its own information about how the message was received, such

複数のViaフィールド値は、メッセージを転送した各プロキシまたはゲートウェイを表します。各中間者はメッセージの受信方法に関する独自の情報を追加し、最終的な結果は転送する受信

that the end result is ordered according to the sequence of forwarding recipients.

者のシーケンスに従って並べられます。

A proxy MUST send an appropriate Via header field, as described below, in each message that it forwards. An HTTP-to-HTTP gateway MUST send an appropriate Via header field in each inbound request message and MAY send a Via header field in forwarded response messages.

プロキシは、転送する各メッセージで、以下に説明するように、適切なViaヘッダーフィールドを送信する必要があります。HTTP-to-HTTPゲートウェイは、各インバウンド要求メッセージで適切なViaヘッダーフィールドを送信する必要があり、転送された応答メッセージでViaヘッダーフィールドを送信する必要があります。

For each intermediary, the received-protocol indicates the protocol and protocol version used by the upstream sender of the message. Hence, the Via field value records the advertised protocol capabilities of the request/response chain such that they remain visible to downstream recipients; this can be useful for determining what backwards-incompatible features might be safe to use in response, or within a later request, as described in Section 2.6. For brevity, the protocol-name is omitted when the received protocol is HTTP.

仲介者ごとに、received-protocolは、メッセージのアップストリーム送信者が使用したプロトコルとプロトコルバージョンを示します。したがって、Viaフィールドの値は、要求/応答チェーンのアドバタイズされたプロトコル機能を記録し、下流の受信者から見える状態を維持します。セクション2.6で説明されているように、これは、下位互換性のない機能が応答で、または後の要求内で安全に使用できる可能性があるものを判断するのに役立ちます。簡潔にするために、受信したプロトコルがHTTPの場合、プロトコル名は省略されます。

The received-by portion of the field value is normally the host and optional port number of a recipient server or client that subsequently forwarded the message. However, if the real host is considered to be sensitive information, a sender MAY replace it with a pseudonym. If a port is not provided, a recipient MAY interpret that as meaning it was received on the default TCP port, if any, for the received-protocol.

フィールド値のreceived-by部分は通常、その後メッセージを転送した受信者サーバーまたはクライアントのホストおよびオプションのポート番号です。ただし、実際のホストが機密情報であると見なされる場合、送信者はそれを仮名に置き換えることができます（MAY）。ポートが提供されていない場合、受信者は、それが受信されたプロトコルのデフォルトのTCPポート（ある場合）で受信されたことを意味すると解釈してもよい（MAY）。

A sender MAY generate comments in the Via header field to identify the software of each recipient, analogous to the User-Agent and Server header fields. However, all comments in the Via field are optional, and a recipient MAY remove them prior to forwarding the message.

送信者は、User-AgentおよびServerヘッダーフィールドと同様に、Viaヘッダーフィールドにコメントを生成して、各受信者のソフトウェアを識別できます。ただし、[Via]フィールドのコメントはすべてオプションであり、受信者はメッセージを転送する前にコメントを削除できます（MAY）。

For example, a request message could be sent from an HTTP/1.0 user agent to an internal proxy code-named "fred", which uses HTTP/1.1 to forward the request to a public proxy at p.example.net, which completes the request by forwarding it to the origin server at www.example.com. The request received by www.example.com would then have the following Via header field:

たとえば、リクエストメッセージはHTTP / 1.0ユーザーエージェントから、コード名「fred」の内部プロキシに送信できます。これはHTTP / 1.1を使用して、リクエストをp.example.netのパブリックプロキシに転送します。 www.example.comのオリジンサーバーに転送してリクエストします。 www.example.comが受け取ったリクエストには、次のViaヘッダーフィールドがあります。

```
Via: 1.0 fred, 1.1 p.example.net
```

```
Via：1.0 fred、1.1 p.example.net
```

An intermediary used as a portal through a network firewall SHOULD NOT forward the names and ports of hosts within the firewall region unless it is explicitly enabled to do so. If not enabled, such an intermediary SHOULD replace each received-by host of any host behind the firewall by an appropriate pseudonym for that host.

ネットワークファイアウォールを介してポータルとして使用される仲介者は、明示的に有効にされていない限り、ファイアウォールリージョン内のホストの名前とポートを転送しないでください。有効になっていない場合、そのような中間のSHOULDは、ファイアウォールの背後にある任意のホストの各受信ホストを、そのホストの適切な仮名に置き換える必要があります（SHOULD）。

An intermediary MAY combine an ordered subsequence of Via header field entries into a single such entry if the entries have identical received-protocol values. For example,

エントリが同一の受信プロトコル値を持っている場合、仲介者は、Viaヘッダーフィールドエントリの順序付けられたサブシーケンスをそのような単一のエントリに結合してもよい（MAY）。例えば、

```
Via: 1.0 ricky, 1.1 ethel, 1.1 fred, 1.0 lucy
```

```
Via：1.0 ricky、1.1 ethel、1.1 fred、1.0 lucy
```

could be collapsed to

に崩壊する可能性があります

```
Via: 1.0 ricky, 1.1 mertz, 1.0 lucy
```

```
Via：1.0 ricky、1.1 mertz、1.0 lucy
```

A sender SHOULD NOT combine multiple entries unless they are all under the same organizational control and the hosts have already been replaced by pseudonyms. A sender MUST

送信者は、すべてが同じ組織の制御下にあり、ホストが仮名に既に置き換えられていない限り、複数のエントリを結合してはなりません（SHOULD NOT）。送信者は、異なる受信プロトコ

NOT combine entries that have different received-protocol values.

ル値を持つエントリを組み合わせてはなりません（MUST NOT）。

5.7.2. Transformations

Some intermediaries include features for transforming messages and their payloads. A proxy might, for example, convert between image formats in order to save cache space or to reduce the amount of traffic on a slow link. However, operational problems might occur when these transformations are applied to payloads intended for critical applications, such as medical imaging or scientific data analysis, particularly when integrity checks or digital signatures are used to ensure that the payload received is identical to the original.

An HTTP-to-HTTP proxy is called a "transforming proxy" if it is designed or configured to modify messages in a semantically meaningful way (i.e., modifications, beyond those required by normal HTTP processing, that change the message in a way that would be significant to the original sender or potentially significant to downstream recipients). For example, a transforming proxy might be acting as a shared annotation server (modifying responses to include references to a local annotation database), a malware filter, a format transcoder, or a privacy filter. Such transformations are presumed to be desired by whichever client (or client organization) selected the proxy.

If a proxy receives a request-target with a host name that is not a fully qualified domain name, it MAY add its own domain to the host name it received when forwarding the request. A proxy MUST NOT change the host name if the request-target contains a fully qualified domain name.

A proxy MUST NOT modify the "absolute-path" and "query" parts of the received request-target when forwarding it to the next inbound server, except as noted above to replace an empty path with "/" or "*".

A proxy MAY modify the message body through application or removal of a transfer coding (Section 4).

A proxy MUST NOT transform the payload (Section 3.3 of [RFC7231]) of a message that contains a no-transform cache-control directive (Section 5.2 of [RFC7234]).

A proxy MAY transform the payload of a message that does not contain a no-transform cache-control directive. A proxy that transforms a payload MUST add a Warning header field with the warn-code of 214 ("Transformation Applied") if one is not already in the message (see Section 5.5 of [RFC7234]). A proxy that transforms the payload of a 200 (OK) response can further inform downstream recipients that a transformation has been applied by changing the response status code to 203 (Non-Authoritative Information) (Section 6.3.4 of [RFC7231]).

A proxy SHOULD NOT modify header fields that provide information about the endpoints of the communication chain, the resource state, or the selected representation (other than the payload) unless the field's definition specifically allows such modification or the modification is deemed necessary for privacy or security.

5.7.2. 変革

一部の仲介者には、メッセージとそのペイロードを変換する機能が含まれています。プロキシは、たとえば、キャッシュスペースを節約したり、低速リンクのトラフィック量を削減したりするために、画像フォーマット間で変換する場合があります。ただし、医用画像処理や科学データ分析などの重要なアプリケーションを対象としたペイロードにこれらの変換を適用すると、特に整合性チェックまたはデジタル署名を使用して受信したペイロードが元のペイロードと同一であることを確認すると、運用上の問題が発生する可能性があります。

HTTPからHTTPへのプロキシは、意味的に意味のある方法でメッセージを変更するように設計または構成されている場合（つまり、通常のHTTP処理で必要なもの以外の変更で、メッセージを変更する方法で「変換プロキシ」と呼ばれます。元の送信者にとって重要であるか、下流の受信者にとって重要である可能性があります）。たとえば、変換プロキシは、共有注釈サーバー（ローカル注釈データベースへの参照を含めるように応答を変更する）、マルウェアフィルター、フォーマットトランスコーダー、またはプライバシーフィルターとして機能している場合があります。このような変換は、プロキシを選択したクライアント（またはクライアント組織）が希望するものと見なされます。

プロキシが、完全修飾ドメイン名ではないホスト名を持つリクエストターゲットを受信した場合、リクエストの転送時に受信したホスト名に独自のドメインを追加できます（MAY）。request-targetに完全修飾ドメイン名が含まれている場合、プロキシはホスト名を変更してはなりません（MUST NOT）。

空のパスを「/」または「*」で置き換えるために上記の場合を除いて、プロキシは、次の受信サーバーに転送するときに、受信した要求ターゲットの「絶対パス」および「クエリ」部分を変更してはなりません（MUST NOT）。

プロキシは、転送コーディングの適用または削除（セクション4）を通じてメッセージ本文を変更できます（MAY）。

プロキシは、変換なしのキャッシュ制御ディレクティブ（[RFC7234]のセクション5.2）を含むメッセージのペイロード（[RFC7231]のセクション3.3）を変換してはなりません（MUST NOT）。

プロキシは、変換なしのキャッシュ制御ディレクティブを含まないメッセージのペイロードを変換してもよい（MAY）。ペイロードを変換するプロキシは、警告ヘッダーフィールドが214（「変換適用済み」）の警告ヘッダーフィールドを追加する必要があります（メッセージ内にない場合（[RFC7234]のセクション5.5を参照））。 200（OK）応答のペイロードを変換するプロキシは、応答ステータスコードを203（非権威情報）に変更することにより、変換が適用されたことを下流の受信者にさらに通知できます（[RFC7231]のセクション6.3.4）。

プロキシは、通信チェーンのエンドポイント、リソースの状態、または選択された表現（ペイロード以外）に関する情報を提供するヘッダーフィールドを変更してはなりません（フィールドの定義でそのような変更が特に許可されている場合、または変更がプライバシーやセキュリティに必要であると見なされている場合を除く）。 。

6. Connection Management

6. 接続管理

HTTP messaging is independent of the underlying transport- or session-layer connection protocol(s). HTTP only presumes a reliable transport with in-order delivery of requests and the corresponding in-order delivery of responses. The mapping of HTTP request and response structures onto the data units of an underlying transport protocol is outside the scope of this specification.

HTTPメッセージングは、基になるトランスポート層またはセッション層の接続プロトコルには依存しません。 HTTPは、要求の順序どおりの配信と対応する応答の順序どおりの配信を伴う信頼性の高いトランスポートのみを前提としています。 HTTP要求および応答構造の、基礎となるトランスポートプロトコルのデータユニットへのマッピングは、この仕様の範囲外です。

As described in Section 5.2, the specific connection protocols to be used for an HTTP interaction are determined by client configuration and the target URI. For example, the "http" URI scheme (Section 2.7.1) indicates a default connection of TCP over IP, with a default TCP port of 80, but the client might be configured to use a proxy via some other connection, port, or protocol.

セクション5.2で説明されているように、HTTP対話に使用される特定の接続プロトコルは、クライアント構成とターゲットURIによって決定されます。たとえば、「http」URIスキーム（セクション2.7.1）は、TCP over IPのデフォルト接続を示し、デフォルトのTCPポートは80ですが、クライアントは他の接続、ポート、またはプロトコル。

HTTP implementations are expected to engage in connection management, which includes maintaining the state of current connections, establishing a new connection or reusing an existing connection, processing messages received on a connection, detecting connection failures, and closing each connection. Most clients maintain multiple connections in parallel, including more than one connection per server endpoint. Most servers are designed to maintain thousands of concurrent connections, while controlling request queues to enable fair use and detect denial-of-service attacks.

HTTP実装は、現在の接続の状態の維持、新しい接続の確立または既存の接続の再利用、接続で受信したメッセージの処理、接続の失敗の検出、各接続のクローズなどの接続管理に従事することが求められます。ほとんどのクライアントは、サーバーエンドポイントごとに複数の接続を含め、複数の接続を並行して維持します。ほとんどのサーバーは、要求キューを制御してフェアユースを可能にし、サービス拒否攻撃を検出する一方で、数千の同時接続を維持するように設計されています。

6.1. Connection

6.1. 接続

The "Connection" header field allows the sender to indicate desired control options for the current connection. In order to avoid confusing downstream recipients, a proxy or gateway MUST remove or replace any received connection options before forwarding the message.

「接続」ヘッダーフィールドを使用すると、送信者は現在の接続に必要な制御オプションを指定できます。ダウンストリーム受信者の混乱を避けるために、プロキシまたはゲートウェイは、メッセージを転送する前に、受信した接続オプションを削除または置き換える必要があります。

When a header field aside from Connection is used to supply control information for or about the current connection, the sender MUST list the corresponding field-name within the Connection header field. A proxy or gateway MUST parse a received Connection header field before a message is forwarded and, for each connection-option in this field, remove any header field(s) from the message with the same name as the connection-option, and then remove the Connection header field itself (or replace it with the intermediary's own connection options for the forwarded message).

Connection以外のヘッダーフィールドを使用して現在の接続の制御情報を提供する場合、送信者は、Connectionヘッダーフィールド内に対応するフィールド名をリストする必要があります。プロキシまたはゲートウェイは、メッセージが転送される前に受信した接続ヘッダーフィールドを解析する必要があり、このフィールドの各接続オプションについて、接続オプションと同じ名前のメッセージからヘッダーフィールドを削除してから削除しますConnectionヘッダーフィールド自体（または転送メッセージの仲介者独自の接続オプションに置き換えます）。

Hence, the Connection header field provides a declarative way of distinguishing header fields that are only intended for the immediate recipient ("hop-by-hop") from those fields that are intended for all recipients on the chain ("end-to-end"), enabling the message to be self-descriptive and allowing future connection-specific extensions to be deployed without fear that they will be blindly forwarded by older intermediaries.

したがって、接続ヘッダーフィールドは、直接の受信者のみを対象とするヘッダーフィールド（「ホップバイホップ」）を、チェーン上のすべての受信者を対象とするフィールド（「エンドツーエンド」）と区別する宣言的な方法を提供します。 "）、メッセージが自己記述的であり、将来の接続固有の拡張が古い仲介者によって盲目的に転送されることを恐れずに展開できるようにします。

The Connection header field's value has the following grammar:

Connectionヘッダーフィールドの値には、次の文法があります。

```
Connection = 1#connection-option
connection-option = token
```

```
接続= 1＃connection-option
接続オプション=トークン
```

Connection options are case-insensitive.

接続オプションは大文字と小文字を区別しません。

A sender MUST NOT send a connection option corresponding to a header field that is intended for all recipients of the payload. For example, Cache-Control is never appropriate as a connection option (Section 5.2 of [RFC7234]).

送信者は、ペイロードのすべての受信者を対象とするヘッダーフィールドに対応する接続オプションを送信してはなりません（MUST NOT）。たとえば、Cache-Controlは接続オプションとして適切ではありません（[RFC7234]のセクション5.2）。

The connection options do not always correspond to a header field present in the message, since a connection-specific header field might not be needed if there are no parameters associated with a connection option. In contrast, a connection-specific header field that is received without a corresponding connection option usually indicates that the field has been improperly forwarded by an intermediary and ought to be ignored by the recipient.

接続オプションに関連付けられたパラメーターがない場合、接続固有のヘッダーフィールドは必要ない可能性があるため、接続オプションは常にメッセージに存在するヘッダーフィールドに対応するとは限りません。対照的に、対応する接続オプションなしで受信された接続固有のヘッダーフィールドは、通常、フィールドが仲介者によって不適切に転送され、受信者によって無視されるべきであることを示します。

When defining new connection options, specification authors ought to survey existing header field names and ensure that the new connection option does not share the same name as an already deployed header field. Defining a new connection option essentially reserves that potential field-name for carrying additional information related to the connection option, since it would be unwise for senders to use that field-name for anything else.

新しい接続オプションを定義する場合、仕様の作成者は既存のヘッダーフィールド名を調査し、新しい接続オプションが既に展開されているヘッダーフィールドと同じ名前を共有しないようにする必要があります。新しい接続オプションを定義すると、送信者が他の目的でそのフィールド名を使用するのは賢明ではないため、接続オプションに関連する追加情報を運ぶために、その潜在的なフィールド名を本質的に予約します。

The "close" connection option is defined for a sender to signal that this connection will be closed after completion of the response. For example,

「クローズ」接続オプションは、送信者が応答の完了後にこの接続がクローズされることを通知するために定義されています。例えば、

Connection: close

接続：閉じる

in either the request or the response header fields indicates that the sender is going to close the connection after the current request/response is complete (Section 6.6).

要求または応答ヘッダーフィールドのどちらかは、現在の要求/応答が完了した後に送信者が接続を閉じることを示します（セクション6.6）。

A client that does not support persistent connections MUST send the "close" connection option in every request message.

永続的な接続をサポートしないクライアントは、すべての要求メッセージで「閉じる」接続オプションを送信する必要があります。

A server that does not support persistent connections MUST send the "close" connection option in every response message that does not have a 1xx (Informational) status code.

持続的接続をサポートしないサーバーは、1xx（情報）ステータスコードを持たないすべての応答メッセージで「閉じる」接続オプションを送信する必要があります。

6.2. Establishment

6.2. 確立

It is beyond the scope of this specification to describe how connections are established via various transport- or session-layer protocols. Each connection applies to only one transport link.

さまざまなトランスポート層またはセッション層プロトコルを介して接続が確立される方法を説明することは、この仕様の範囲を超えています。各接続は、1つのトランスポートリンクにのみ適用されます。

6.3. Persistence

6.3. 持続性

HTTP/1.1 defaults to the use of "persistent connections", allowing multiple requests and responses to be carried over a single connection. The "close" connection option is used to signal that a connection will not persist after the current request/response. HTTP implementations SHOULD support persistent connections.

HTTP / 1.1はデフォルトで「持続的接続」を使用するため、複数の要求と応答を単一の接続で伝送できます。 「閉じる」接続オプションは、現在の要求/応答後に接続が持続しないことを通知するために使用されます。 HTTP実装は永続的な接続をサポートする必要があります（SHOULD）。

A recipient determines whether a connection is persistent or not based on the most recently received message's protocol version and Connection header field (if any):

受信者は、最後に受信したメッセージのプロトコルバージョンと接続ヘッダーフィールド（存在する場合）に基づいて、接続が持続的かどうかを判断します。

o If the "close" connection option is present, the connection will not persist after the current response; else,

o 「閉じる」接続オプションが存在する場合、現在の応答後に接続は持続しません。そうしないと、

o If the received protocol is HTTP/1.1 (or later), the connection will persist after the current response; else,

o 受信したプロトコルがHTTP / 1.1以降の場合、現在の応答後も接続は維持されます。そうしないと、

o If the received protocol is HTTP/1.0, the "keep-alive" connection option is present, the recipient is not a proxy, and the recipient wishes to honor the HTTP/1.0 "keep-alive" mechanism, the connection will persist after the current response; otherwise,

o 受信したプロトコルがHTTP / 1.0で、「キープアライブ」接続オプションが存在し、受信者がプロキシではなく、受信者がHTTP / 1.0の「キープアライブ」メカニズムを尊重したい場合、接続は現在の応答。さもないと、

o The connection will close after the current response.

A client MAY send additional requests on a persistent connection until it sends or receives a "close" connection option or receives an HTTP/1.0 response without a "keep-alive" connection option.

In order to remain persistent, all messages on a connection need to have a self-defined message length (i.e., one not defined by closure of the connection), as described in Section 3.3. A server MUST read the entire request message body or close the connection after sending its response, since otherwise the remaining data on a persistent connection would be misinterpreted as the next request. Likewise, a client MUST read the entire response message body if it intends to reuse the same connection for a subsequent request.

A proxy server MUST NOT maintain a persistent connection with an HTTP/1.0 client (see Section 19.7.1 of [RFC2068] for information and discussion of the problems with the Keep-Alive header field implemented by many HTTP/1.0 clients).

See Appendix A.1.2 for more information on backwards compatibility with HTTP/1.0 clients.

6.3.1. Retrying Requests

Connections can be closed at any time, with or without intention. Implementations ought to anticipate the need to recover from asynchronous close events.

When an inbound connection is closed prematurely, a client MAY open a new connection and automatically retransmit an aborted sequence of requests if all of those requests have idempotent methods (Section 4.2.2 of [RFC7231]). A proxy MUST NOT automatically retry non-idempotent requests.

A user agent MUST NOT automatically retry a request with a non-idempotent method unless it has some means to know that the request semantics are actually idempotent, regardless of the method, or some means to detect that the original request was never applied. For example, a user agent that knows (through design or configuration) that a POST request to a given resource is safe can repeat that request automatically. Likewise, a user agent designed specifically to operate on a version control repository might be able to recover from partial failure conditions by checking the target resource revision(s) after a failed connection, reverting or fixing any changes that were partially applied, and then automatically retrying the requests that failed.

A client SHOULD NOT automatically retry a failed automatic retry.

6.3.2. Pipelining

A client that supports persistent connections MAY "pipeline" its requests (i.e., send multiple requests without waiting for each response). A server MAY process a sequence of pipelined requests in parallel if they all have safe methods (Section 4.2.1 of [RFC7231]), but it MUST send the corresponding responses in the same order that the requests were received.

A client that pipelines requests SHOULD retry unanswered requests if the connection closes before it receives all of the corresponding responses. When retrying pipelined requests

o 現在の応答の後で接続が閉じます。

クライアントは、「閉じる」接続オプションを送信または受信するか、「キープアライブ」接続オプションなしでHTTP / 1.0応答を受信するまで、永続的な接続で追加の要求を送信できます（MAY）。

セクション3.3で説明されているように、永続性を維持するには、接続上のすべてのメッセージに自己定義のメッセージ長（つまり、接続のクローズによって定義されていないメッセージ長）が必要です。サーバーは、要求メッセージ本文全体を読み取るか、応答を送信した後で接続を閉じる必要があります。そうしないと、永続的な接続の残りのデータが次の要求として誤って解釈されます。同様に、クライアントは、後続のリクエストで同じ接続を再利用する場合は、応答メッセージ本文全体を読み取る必要があります。

プロキシサーバーは、HTTP / 1.0クライアントとの永続的な接続を維持してはなりません（多くのHTTP / 1.0クライアントによって実装されたKeep-Aliveヘッダーフィールドの問題と情報については、[RFC2068]のセクション19.7.1を参照してください）。

HTTP / 1.0クライアントとの下位互換性の詳細については、付録A.1.2を参照してください。

6.3.1. リクエストの再試行

接続は、意図的または非意図的にいつでも閉じることができます。実装では、非同期のクローズイベントから回復する必要性を予測する必要があります。

インバウンド接続が時期尚早に閉じられると、クライアントは新しい接続を開いて、それらのすべての要求がべき等メソッドを持っている場合、中止された要求のシーケンスを自動的に再送信できます（[RFC7231]のセクション4.2.2）。プロキシは、べき等でない要求を自動的に再試行してはなりません。

ユーザーエージェントは、メソッドに関係なくリクエストのセマンティクスが実際にべき等であることを知る手段、または元のリクエストが適用されなかったことを検出する手段がない限り、非べき等メソッドでリクエストを自動的に再試行してはなりません。たとえば、特定のリソースへのPOST要求が安全であることを（設計または構成を通じて）知っているユーザーエージェントは、その要求を自動的に繰り返すことができます。同様に、バージョンコントロールリポジトリを操作するように特別に設計されたユーザーエージェントは、接続の失敗後にターゲットリソースのリビジョンを確認し、部分的に適用された変更を元に戻すか修正することで、部分的な障害状態から回復できる場合があります。失敗した要求を再試行します。

クライアントは、失敗した自動再試行を自動的に再試行すべきではありません（SHOULD NOT）。

6.3.2. パイプライン

永続的な接続をサポートするクライアントは、そのリクエストを「パイプライン化」することができます（つまり、各応答を待たずに複数のリクエストを送信できます）。サーバーは、パイプライン化された一連のリクエストを安全なメソッド（[RFC7231]のセクション4.2.1）がある場合、並行して処理できますが、対応する応答を、リクエストが受信されたのと同じ順序で送信する必要があります。

リクエストをパイプライン処理するクライアントは、対応するすべての応答を受信する前に接続が閉じた場合、未応答のリクエストを再試行する必要があります（SHOULD）。失敗した接

after a failed connection (a connection not explicitly closed by the server in its last complete response), a client MUST NOT pipeline immediately after connection establishment, since the first remaining request in the prior pipeline might have caused an error response that can be lost again if multiple requests are sent on a prematurely closed connection (see the TCP reset problem described in Section 6.6).

続（最後の完全な応答でサーバーによって明示的に閉じられていない接続）の後にパイプライン要求を再試行する場合、クライアントは接続確立の直後にパイプラインを実行してはなりません。途中で閉じられた接続で複数の要求が送信されると、再び失われる可能性があります（6.6節で説明されているTCPリセットの問題を参照）。

Idempotent methods (Section 4.2.2 of [RFC7231]) are significant to pipelining because they can be automatically retried after a connection failure. A user agent SHOULD NOT pipeline requests after a non-idempotent method, until the final response status code for that method has been received, unless the user agent has a means to detect and recover from partial failure conditions involving the pipelined sequence.

べき等メソッド（[RFC7231]のセクション4.2.2）は、接続障害の後に自動的に再試行できるため、パイプライン化にとって重要です。ユーザーエージェントがパイプラインシーケンスを含む部分的な障害状態を検出して回復する手段を備えていない限り、非べき等メソッドの後、そのメソッドの最終応答ステータスコードが受信されるまで、ユーザーエージェントはリクエストをパイプライン処理すべきではありません。

An intermediary that receives pipelined requests MAY pipeline those requests when forwarding them inbound, since it can rely on the outbound user agent(s) to determine what requests can be safely pipelined. If the inbound connection fails before receiving a response, the pipelining intermediary MAY attempt to retry a sequence of requests that have yet to receive a response if the requests all have idempotent methods; otherwise, the pipelining intermediary SHOULD forward any received responses and then close the corresponding outbound connection(s) so that the outbound user agent(s) can recover accordingly.

パイプライン化されたリクエストを受信する仲介者は、それらのリクエストをインバウンドで転送するときにパイプライン化できます。これは、アウトバウンドのユーザーエージェントに依存して、どのリクエストを安全にパイプライン化できるかを判断できるためです。応答を受信する前にインバウンド接続が失敗した場合、パイプライン処理中間手段は、すべての要求にべき等メソッドがある場合、応答をまだ受信していない一連の要求を再試行する場合があります。それ以外の場合、パイプライン処理の仲介者は、受信した応答を転送し、対応する送信接続を閉じて、送信ユーザーエージェントが適切に回復できるようにする必要があります。

6.4. Concurrency

6.4. 並行性

A client ought to limit the number of simultaneous open connections that it maintains to a given server.

クライアントは、特定のサーバーに対して維持する同時オープン接続の数を制限する必要があります。

Previous revisions of HTTP gave a specific number of connections as a ceiling, but this was found to be impractical for many applications. As a result, this specification does not mandate a particular maximum number of connections but, instead, encourages clients to be conservative when opening multiple connections.

HTTPの以前のリビジョンでは、上限として特定の数の接続が提供されていましたが、これは多くのアプリケーションにとって非実用的であることが判明しました。その結果、この仕様は特定の最大接続数を義務付けていませんが、代わりに、複数の接続を開くときにクライアントを保守的にすることを推奨しています。

Multiple connections are typically used to avoid the "head-of-line blocking" problem, wherein a request that takes significant server-side processing and/or has a large payload blocks subsequent requests on the same connection. However, each connection consumes server resources. Furthermore, using multiple connections can cause undesirable side effects in congested networks.

複数の接続は通常、「ヘッドオブラインブロッキング」の問題を回避するために使用されます。この場合、サーバー側の重要な処理や大きなペイロードを持つ要求は、同じ接続での後続の要求をブロックします。ただし、接続ごとにサーバーリソースが消費されます。さらに、複数の接続を使用すると、輻輳したネットワークで望ましくない副作用が発生する可能性があります。

Note that a server might reject traffic that it deems abusive or characteristic of a denial-of-service attack, such as an excessive number of open connections from a single client.

サーバーは、1つのクライアントからのオープン接続の数が多すぎるなど、悪意がある、またはサービス拒否攻撃の特徴であると見なすトラフィックを拒否する場合があることに注意してください。

6.5. Failures and Timeouts

6.5. 失敗とタイムアウト

Servers will usually have some timeout value beyond which they will no longer maintain an inactive connection. Proxy servers might make this a higher value since it is likely that the client will be making more connections through the same proxy server. The use of persistent connections places no requirements on the length (or existence) of this timeout for either the client or the server.

サーバーには通常、タイムアウト値があり、それを超えると非アクティブな接続を維持できなくなります。クライアントが同じプロキシサーバーを介してより多くの接続を行う可能性が高いため、プロキシサーバーはこれをより高い値にする場合があります。永続的な接続を使用しても、クライアントまたはサーバーのいずれかでこのタイムアウトの長さ（または存在）に要件はありません。

A client or server that wishes to time out SHOULD issue a graceful close on the connection. Implementations SHOULD constantly monitor open connections for a received closure

タイムアウトを希望するクライアントまたはサーバーは、接続に対して適切なクローズを発行する必要があります（SHOULD）。接続の両側を即座に閉じると、割り当てられた

signal and respond to it as appropriate, since prompt closure of both sides of a connection enables allocated system resources to be reclaimed.

A client, server, or proxy MAY close the transport connection at any time. For example, a client might have started to send a new request at the same time that the server has decided to close the "idle" connection. From the server's point of view, the connection is being closed while it was idle, but from the client's point of view, a request is in progress.

A server SHOULD sustain persistent connections, when possible, and allow the underlying transport's flow-control mechanisms to resolve temporary overloads, rather than terminate connections with the expectation that clients will retry. The latter technique can exacerbate network congestion.

A client sending a message body SHOULD monitor the network connection for an error response while it is transmitting the request. If the client sees a response that indicates the server does not wish to receive the message body and is closing the connection, the client SHOULD immediately cease transmitting the body and close its side of the connection.

システムリソースを再利用できるため、実装では、開いている接続が受信した閉鎖信号を常に監視し、適切に応答する必要があります（SHOULD）。

クライアント、サーバー、またはプロキシは、いつでもトランスポート接続を閉じることができます。たとえば、サーバーが「アイドル」接続を閉じることを決定したと同時に、クライアントが新しい要求の送信を開始した可能性があります。サーバーの観点からは、接続はアイドル状態のときに閉じられていますが、クライアントの観点からは、要求が進行中です。

サーバーは、可能であれば永続的な接続を維持し、クライアントが再試行することを想定して接続を終了するのではなく、基になるトランスポートのフロー制御メカニズムが一時的な過負荷を解決できるようにする必要があります。後者の手法は、ネットワークの輻輳を悪化させる可能性があります。

メッセージ本文を送信するクライアントは、リクエストを送信している間、エラー応答についてネットワーク接続を監視する必要があります（SHOULD）。サーバーがメッセージ本文を受信したくないことを示す応答をクライアントが確認し、接続を閉じている場合、クライアントは本文の送信を直ちに中止し、接続の側面を閉じる必要があります（SHOULD）。

6.6. Tear-down

The Connection header field (Section 6.1) provides a "close" connection option that a sender SHOULD send when it wishes to close the connection after the current request/response pair.

A client that sends a "close" connection option MUST NOT send further requests on that connection (after the one containing "close") and MUST close the connection after reading the final response message corresponding to this request.

A server that receives a "close" connection option MUST initiate a close of the connection (see below) after it sends the final response to the request that contained "close". The server SHOULD send a "close" connection option in its final response on that connection. The server MUST NOT process any further requests received on that connection.

A server that sends a "close" connection option MUST initiate a close of the connection (see below) after it sends the response containing "close". The server MUST NOT process any further requests received on that connection.

A client that receives a "close" connection option MUST cease sending requests on that connection and close the connection after reading the response message containing the "close"; if additional pipelined requests had been sent on the connection, the client SHOULD NOT assume that they will be processed by the server.

If a server performs an immediate close of a TCP connection, there is a significant risk that the client will not be able to read the last HTTP response. If the server receives additional data from the client on a fully closed connection, such as another request that was sent by the client before receiving the server's response, the server's TCP stack will send a reset packet to the client; unfortunately, the reset packet might erase the client's unacknowledged input buffers before they can be read and interpreted by the client's HTTP parser.

To avoid the TCP reset problem, servers typically close a

6.6. 取り壊す

Connectionヘッダーフィールド（6.1節）は、現在の要求/応答ペアの後に接続を閉じたい場合に送信者が送信する必要がある「閉じる」接続オプションを提供します。

「閉じる」接続オプションを送信するクライアントは、その接続で（「閉じる」を含むものの後に）それ以上の要求を送信してはならず（MUST）、この要求に対応する最終応答メッセージを読んだ後に接続を閉じなければなりません。

「閉じる」接続オプションを受信したサーバーは、「閉じる」を含むリクエストへの最終応答を送信した後、接続の終了（下記参照）を開始する必要があります。サーバーは、その接続の最終応答で「クローズ」接続オプションを送信する必要があります（SHOULD）。サーバーは、その接続で受信したそれ以上の要求を処理してはなりません（MUST NOT）。

「閉じる」接続オプションを送信するサーバーは、「閉じる」を含む応答を送信した後、接続の終了（以下を参照）を開始する必要があります。サーバーは、その接続で受信したそれ以上の要求を処理してはなりません（MUST NOT）。

「閉じる」接続オプションを受信したクライアントは、その接続での要求の送信を中止し、「閉じる」を含む応答メッセージを読み取った後に接続を閉じる必要があります。追加のパイプライン化された要求が接続で送信された場合、クライアントはそれらがサーバーによって処理されることを想定してはなりません（SHOULD NOT）。

サーバーがTCP接続の即時クローズを実行する場合、クライアントが最後のHTTP応答を読み取ることができないという重大なリスクがあります。サーバーがサーバーからの応答を受信する前にクライアントから送信された別の要求など、完全に閉じた接続でクライアントから追加のデータを受信した場合、サーバーのTCPスタックはリセットパケットをクライアントに送信します。残念ながら、リセットパケットは、クライアントのHTTPパーサーによって読み取られ解釈される前に、クライアントの未確認の入力バッファーを消去する可能性があります。

TCPリセットの問題を回避するために、サーバーは通常、段階

connection in stages. First, the server performs a half-close by closing only the write side of the read/write connection. The server then continues to read from the connection until it receives a corresponding close by the client, or until the server is reasonably certain that its own TCP stack has received the client's acknowledgement of the packet(s) containing the server's last response. Finally, the server fully closes the connection.

的に接続を閉じます。まず、サーバーは読み取り/書き込み接続の書き込み側のみを閉じることにより、ハーフクローズを実行します。その後、サーバーは、クライアントによる対応するクローズを受信するまで、またはサーバーが自身のTCPスタックがサーバーの最後の応答を含むパケットのクライアントの確認応答を受信したことを合理的に確信するまで、接続からの読み取りを続けます。最後に、サーバーは接続を完全に閉じます。

It is unknown whether the reset problem is exclusive to TCP or might also be found in other transport connection protocols.

リセットの問題がTCPに限定されているのか、それとも他のトランスポート接続プロトコルにも見られるのかは不明です。

6.7. Upgrade

6.7. アップグレードする

The "Upgrade" header field is intended to provide a simple mechanism for transitioning from HTTP/1.1 to some other protocol on the same connection. A client MAY send a list of protocols in the Upgrade header field of a request to invite the server to switch to one or more of those protocols, in order of descending preference, before sending the final response. A server MAY ignore a received Upgrade header field if it wishes to continue using the current protocol on that connection. Upgrade cannot be used to insist on a protocol change.

「Upgrade」ヘッダーフィールドは、同じ接続でHTTP / 1.1から他のプロトコルに移行するための簡単なメカニズムを提供することを目的としています。クライアントは、リクエストのUpgradeヘッダーフィールドにプロトコルのリストを送信して、最終的な応答を送信する前に、サーバーをそれらのプロトコルの1つ以上に切り替えるように招待することができます（優先順の降順）。サーバーは、その接続で現在のプロトコルを引き続き使用したい場合は、受信したアップグレードヘッダーフィールドを無視できます。プロトコルの変更を主張するためにアップグレードを使用することはできません。

Upgrade = 1#protocol

アップグレード= 1＃protocol

protocol = protocol-name ["/" protocol-version] protocol-name = token protocol-version = token

protocol = protocol-name ["/" protocol-version] protocol-name = token protocol-version = token

A server that sends a 101 (Switching Protocols) response MUST send an Upgrade header field to indicate the new protocol(s) to which the connection is being switched; if multiple protocol layers are being switched, the sender MUST list the protocols in layer-ascending order. A server MUST NOT switch to a protocol that was not indicated by the client in the corresponding request's Upgrade header field. A server MAY choose to ignore the order of preference indicated by the client and select the new protocol(s) based on other factors, such as the nature of the request or the current load on the server.

101（Switching Protocols）応答を送信するサーバーは、接続の切り替え先の新しいプロトコルを示すためにUpgradeヘッダーフィールドを送信する必要があります。複数のプロトコル層が切り替えられる場合、送信者は層昇順でプロトコルをリストしなければなりません（MUST）。サーバーは、対応する要求のUpgradeヘッダーフィールドでクライアントによって示されていないプロトコルに切り替えてはなりません（MUST NOT）。サーバーは、クライアントによって示された優先順位を無視し、リクエストの性質やサーバーの現在の負荷などの他の要因に基づいて新しいプロトコルを選択することを選択できます（MAY）。

A server that sends a 426 (Upgrade Required) response MUST send an Upgrade header field to indicate the acceptable protocols, in order of descending preference.

426（Upgrade Required）応答を送信するサーバーは、優先度の高い順に、受け入れ可能なプロトコルを示すUpgradeヘッダーフィールドを送信する必要があります。

A server MAY send an Upgrade header field in any other response to advertise that it implements support for upgrading to the listed protocols, in order of descending preference, when appropriate for a future request.

サーバーは、他の応答でUpgradeヘッダーフィールドを送信して、将来のリクエストに適切な場合、降順の優先順で、リストされているプロトコルへのアップグレードのサポートを実装していることをアドバタイズできます（MAY）。

The following is a hypothetical example sent by a client:

以下は、クライアントから送信された架空の例です。

```
    GET /hello.txt HTTP/1.1
    Host: www.example.com
    Connection: upgrade
    Upgrade: HTTP/2.0, SHTTP/1.3, IRC/6.9, RTA/x11
```

The capabilities and nature of the application-level communication after the protocol change is entirely dependent upon the new protocol(s) chosen. However, immediately after sending the 101 (Switching Protocols) response, the server is expected to continue responding to the original request as if it had received its equivalent within the new protocol (i.e., the

プロトコル変更後のアプリケーションレベルの通信の機能と性質は、選択した新しいプロトコルに完全に依存します。ただし、101（Switching Protocols）応答を送信した直後、サーバーは新しいプロトコル内で同等のものを受信したかのように元の要求に応答し続けることが期待されます（つまり、サーバーはプロトコルの後に満たす必要のある未解決の要求をまだ持っています）変更されており、リクエストを繰り返すことなく変更できることが期待されています）。

server still has an outstanding request to satisfy after the protocol has been changed, and is expected to do so without requiring the request to be repeated).

For example, if the Upgrade header field is received in a GET request and the server decides to switch protocols, it first responds with a 101 (Switching Protocols) message in HTTP/1.1 and then immediately follows that with the new protocol's equivalent of a response to a GET on the target resource. This allows a connection to be upgraded to protocols with the same semantics as HTTP without the latency cost of an additional round trip. A server MUST NOT switch protocols unless the received message semantics can be honored by the new protocol; an OPTIONS request can be honored by any protocol.

The following is an example response to the above hypothetical request:

```
HTTP/1.1 101 Switching Protocols Connection: upgrade
Upgrade: HTTP/2.0
[... data stream switches to HTTP/2.0 with an appropriate
response (as defined by new protocol) to the "GET /hello.txt"
request ...]
```

When Upgrade is sent, the sender MUST also send a Connection header field (Section 6.1) that contains an "upgrade" connection option, in order to prevent Upgrade from being accidentally forwarded by intermediaries that might not implement the listed protocols. A server MUST ignore an Upgrade header field that is received in an HTTP/1.0 request.

A client cannot begin using an upgraded protocol on the connection until it has completely sent the request message (i.e., the client can't change the protocol it is sending in the middle of a message). If a server receives both an Upgrade and an Expect header field with the "100-continue" expectation (Section 5.1.1 of [RFC7231]), the server MUST send a 100 (Continue) response before sending a 101 (Switching Protocols) response.

The Upgrade header field only applies to switching protocols on top of the existing connection; it cannot be used to switch the underlying connection (transport) protocol, nor to switch the existing communication to a different connection. For those purposes, it is more appropriate to use a 3xx (Redirection) response (Section 6.4 of [RFC7231]).

This specification only defines the protocol name "HTTP" for use by the family of Hypertext Transfer Protocols, as defined by the HTTP version rules of Section 2.6 and future updates to this specification. Additional tokens ought to be registered with IANA using the registration procedure defined in Section 8.6.

7. ABNF List Extension: #rule

A #rule extension to the ABNF rules of [RFC5234] is used to improve readability in the definitions of some header field values.

A construct "#" is defined, similar to "*", for defining comma-delimited lists of elements. The full form is "<n>#<m>element" indicating at least <n> and at most <m> elements, each

たとえば、GETリクエストでUpgradeヘッダーフィールドが受信され、サーバーがプロトコルを切り替えることを決定した場合、最初にHTTP / 1.1の101（Switching Protocols）メッセージで応答し、その後すぐに新しいプロトコルに相当する応答で応答します。ターゲットリソースのGETに。これにより、接続をHTTPと同じセマンティクスを持つプロトコルにアップグレードして、追加のラウンドトリップの遅延コストを発生させることがなくなります。受信したメッセージのセマンティクスが新しいプロトコルで受け入れられない限り、サーバーはプロトコルを切り替えてはなりません（MUST NOT）。 OPTIONS要求は、どのプロトコルでも受け入れることができます。

以下は、上記の架空の要求に対する応答の例です。

```
HTTP / 1.1 101スイッチングプロトコル接続：アップグレードア
ップグレード：HTTP / 2.0
[...データストリームは、 "GET /hello.txt"リクエストに対する適
切な応答（新しいプロトコルで定義されている）でHTTP / 2.0
に切り替わります...]
```

アップグレードが送信されるとき、送信者は、「アップグレード」接続オプションを含む接続ヘッダーフィールド（6.1節）も送信する必要があります。これにより、リストされているプロトコルを実装していない可能性のある仲介者によってアップグレードが誤って転送されるのを防ぎます。サーバーは、HTTP / 1.0リクエストで受信したUpgradeヘッダーフィールドを無視する必要があります。

クライアントは、要求メッセージを完全に送信するまで、接続でアップグレードされたプロトコルの使用を開始できません（つまり、クライアントは、メッセージの途中で送信しているプロトコルを変更できません）。サーバーが "100-continue"期待値（[RFC7231]のセクション5.1.1）を含むUpgradeおよびExpectヘッダーフィールドの両方を受信した場合、サーバーは101（Switching Protocols）応答を送信する前に100（Continue）応答を送信する必要があります。 。

Upgradeヘッダーフィールドは、既存の接続上のスイッチングプロトコルにのみ適用されます。基になる接続（トランスポート）プロトコルの切り替えや、既存の通信の別の接続への切り替えには使用できません。そのためには、3xx（リダイレクト）応答を使用する方が適切です（[RFC7231]のセクション6.4）。

この仕様では、セクション2.6のHTTPバージョンルールとこの仕様の将来の更新で定義されているように、ハイパーテキスト転送プロトコルのファミリで使用するプロトコル名「HTTP」のみを定義しています。追加のトークンは、セクション8.6で定義された登録手順を使用してIANAに登録する必要があります。

7. ABNFリスト拡張：#rule

[RFC5234]のABNFルールへの#rule拡張は、一部のヘッダーフィールド値の定義を読みやすくするために使用されます。

コンマで区切られた要素のリストを定義するために、「*」と同様の構成「＃」が定義されています。完全な形式は "<n>＃<m> element"であり、少なくとも1つのコンマ（ "、"）とオプション

separated by a single comma (",") and optional whitespace (OWS).

の空白（OWS）で区切られた、少なくとも<n>要素と最大で<m>要素を示します。

In any production that uses the list construct, a sender MUST NOT generate empty list elements. In other words, a sender MUST generate lists that satisfy the following syntax:

リストコンストラクトを使用するプロダクションでは、送信者は空のリスト要素を生成してはなりません（MUST NOT）。つまり、送信者は次の構文を満たすリストを生成する必要があります。

```
1#element => element *( OWS "," OWS element )
```

and:

そして：

```
#element => [ 1#element ]
```

and for n >= 1 and m > 1:

n> = 1およびm> 1の場合：

```
<n>#<m>element => element <n-1>*<m-1>( OWS "," OWS element )
```

For compatibility with legacy list rules, a recipient MUST parse and ignore a reasonable number of empty list elements: enough to handle common mistakes by senders that merge values, but not so much that they could be used as a denial-of-service mechanism. In other words, a recipient MUST accept lists that satisfy the following syntax:

レガシーリストルールとの互換性のために、受信者は妥当な数の空のリスト要素を解析して無視する必要があります。つまり、受信者は次の構文を満たすリストを受け入れる必要があります。

```
#element => [ ( "," / element ) *( OWS "," [ OWS element ] ) ]
```

```
1#element => *( "," OWS ) element *( OWS "," [ OWS element ] )
```

Empty elements do not contribute to the count of elements present. For example, given these ABNF productions:

空の要素は、存在する要素の数には影響しません。たとえば、次のABNFプロダクションがあるとします。

```
example-list      = 1#example-list-elmt
example-list-elmt = token ; see Section 3.2.6
```

Then the following are valid values for example-list (not including the double quotes, which are present for delimitation only):

次に、次はexample-listの有効な値です（区切り文字としてのみ存在する二重引用符は含まれません）。

"foo,bar" "foo ,bar," "foo , ,bar,charlie "

"foo、bar" "foo、bar、" "foo、、bar、charlie"

In contrast, the following values would be invalid, since at least one non-empty element is required by the example-list production:

対照的に、example-listの生成には少なくとも1つの空でない要素が必要であるため、次の値は無効になります。

"" "," ",,"

”” ”、” ”、、”

Appendix B shows the collected ABNF for recipients after the list constructs have been expanded.

付録Bは、リスト構造が拡張された後の受信者用に収集されたABNFを示しています。

8. IANA Considerations

8. IANAに関する考慮事項

8.1. Header Field Registration

8.1. ヘッダーフィールドの登録

HTTP header fields are registered within the "Message Headers" registry maintained at <http://www.iana.org/assignments/message-headers/>.

HTTPヘッダーフィールドは、<http://www.iana.org/assignments/message-headers/>で管理されている「メッセージヘッダー」レジストリ内に登録されます。

This document defines the following HTTP header fields, so the "Permanent Message Header Field Names" registry has been updated accordingly (see [BCP90]).

このドキュメントでは、次のHTTPヘッダーフィールドを定義しているため、「Permanent Message Header Field Names」レジストリはそれに応じて更新されています（[BCP90]を参照）。

```
+-------------------+----------+----------+---------------+
| Header Field Name | Protocol | Status   | Reference     |
+-------------------+----------+----------+---------------+
| Connection        | http     | standard | Section 6.1   |
| Content-Length    | http     | standard | Section 3.3.2 |
```

```
| Host              | http     | standard | Section 5.4   |
| TE                | http     | standard | Section 4.3   |
| Trailer           | http     | standard | Section 4.4   |
| Transfer-Encoding | http     | standard | Section 3.3.1 |
| Upgrade           | http     | standard | Section 6.7   |
| Via               | http     | standard | Section 5.7.1 |
+-------------------+----------+----------+---------------+
```

Furthermore, the header field-name "Close" has been registered as "reserved", since using that name as an HTTP header field might conflict with the "close" connection option of the Connection header field (Section 6.1).

さらに、HTTPヘッダーフィールドとしてその名前を使用すると、接続ヘッダーフィールドの「閉じる」接続オプションと競合する可能性があるため（セクション6.1）、ヘッダーフィールド名「Close」は「予約済み」として登録されています。

```
+-------------------+----------+----------+-------------+
| Header Field Name | Protocol | Status   | Reference   |
+-------------------+----------+----------+-------------+
| Close             | http     | reserved | Section 8.1 |
+-------------------+----------+----------+-------------+
```

The change controller is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

変更管理者は、「IETF（iesg@ietf.org）-Internet Engineering Task Force」です。

8.2. URI Scheme Registration

IANA maintains the registry of URI Schemes [BCP115] at <http://www.iana.org/assignments/uri-schemes/>.

This document defines the following URI schemes, so the "Permanent URI Schemes" registry has been updated accordingly.

8.2. URIスキームの登録

IANAは、URIスキーマ[BCP115]のレジストリを<http://www.iana.org/assignments/uri-schemes/>に保持しています。

このドキュメントでは次のURIスキームを定義しているため、「Permanent URI Schemes」レジストリはそれに応じて更新されています。

```
+------------+-----------------------------------+---------------+
| URI Scheme | Description                       | Reference     |
+------------+-----------------------------------+---------------+
| http       | Hypertext Transfer Protocol       | Section 2.7.1 |
| https      | Hypertext Transfer Protocol Secure | Section 2.7.2 |
+------------+-----------------------------------+---------------+
```

8.3. Internet Media Type Registration

IANA maintains the registry of Internet media types [BCP13] at <http://www.iana.org/assignments/media-types>.

This document serves as the specification for the Internet media types "message/http" and "application/http". The following has been registered with IANA.

8.3. インターネットメディアタイプの登録

IANAは、インターネットメディアタイプのレジストリ[BCP13]を<http://www.iana.org/assignments/media-types>で管理しています。

このドキュメントは、インターネットメディアタイプ「message / http」および「application / http」の仕様として機能します。以下はIANAに登録されています。

8.3.1. Internet Media Type message/http

The message/http type can be used to enclose a single HTTP request or response message, provided that it obeys the MIME restrictions for all "message" types regarding line length and encodings.

Type name: message

Subtype name: http

Required parameters: N/A

Optional parameters: version, msgtype

version: The HTTP-version number of the enclosed message (e.g., "1.1"). If not present, the version can be determined from the first line of the body.

msgtype: The message type -- "request" or "response". If not present, the type can be determined from the first line of the

8.3.1. インターネットメディアタイプメッセージ/http

メッセージ/ httpタイプは、行の長さとエンコーディングに関するすべての「メッセージ」タイプのMIME制限に従う場合、単一のHTTP要求または応答メッセージを囲むために使用できます。

タイプ名：メッセージ

サブタイプ名：http

必須パラメーター：なし

オプションのパラメーター：version、msgtype

version：囲まれたメッセージのHTTPバージョン番号（例：「1.1」）。存在しない場合、バージョンは本文の最初の行から判断できます。

msgtype：メッセージタイプ-「リクエスト」または「レスポンス」。存在しない場合、タイプは本文の最初の行から判別でき

body.

ます。

Encoding considerations: only "7bit", "8bit", or "binary" are permitted

エンコードに関する考慮事項：「7ビット」、「8ビット」、または「バイナリ」のみが許可されます

Security considerations: see Section 9

セキュリティに関する考慮事項：セクション9を参照

Interoperability considerations: N/A

相互運用性に関する考慮事項：N / A

Published specification: This specification (see Section 8.3.1).

公開された仕様：この仕様（セクション8.3.1を参照）。

Applications that use this media type: N/A

このメディアタイプを使用するアプリケーション：N / A

Fragment identifier considerations: N/A

フラグメント識別子の考慮事項：なし

Additional information:

追加情報：

```
Magic number(s):  N/A
```

Deprecated alias names for this type: N/A

このタイプの非推奨のエイリアス名：N / A

```
File extension(s):  N/A
```

```
Macintosh file type code(s):  N/A
```

Person and email address to contact for further information: See Authors' Addresses section.

詳細について連絡する人とメールアドレス：作者のアドレスセクションを参照してください。

Intended usage: COMMON

使用目的：COMMON

Restrictions on usage: N/A

使用上の制限：N / A

Author: See Authors' Addresses section.

作成者：「作成者のアドレス」セクションを参照してください。

Change controller: IESG

コントローラーの変更：IESG

8.3.2. Internet Media Type application/http

8.3.2. インターネットメディアタイプアプリケーション/ http

The application/http type can be used to enclose a pipeline of one or more HTTP request or response messages (not intermixed).

application / httpタイプは、1つ以上のHTTP要求または応答メッセージ（混合されていない）のパイプラインを囲むために使用できます。

Type name: application

タイプ名：アプリケーション

Subtype name: http

サブタイプ名：http

Required parameters: N/A

必須パラメーター：なし

Optional parameters: version, msgtype

オプションのパラメーター：version、msgtype

version: The HTTP-version number of the enclosed messages (e.g., "1.1"). If not present, the version can be determined from the first line of the body.

version：囲まれたメッセージのHTTPバージョン番号（例：「1.1」）。存在しない場合、バージョンは本文の最初の行から判断できます。

msgtype: The message type -- "request" or "response". If not present, the type can be determined from the first line of the body.

msgtype：メッセージタイプ-「リクエスト」または「レスポンス」。存在しない場合、タイプは本文の最初の行から判別できます。

Encoding considerations: HTTP messages enclosed by this type are in "binary" format; use of an appropriate Content-Transfer-Encoding is required when transmitted via email.

エンコードに関する考慮事項：このタイプで囲まれたHTTPメッセージは「バイナリ」形式です。電子メールで送信する場合は、適切なContent-Transfer-Encodingを使用する必要があります。

Security considerations: see Section 9

セキュリティに関する考慮事項：セクション9を参照

Interoperability considerations: N/A

相互運用性に関する考慮事項：N / A

Published specification: This specification (see Section 8.3.2).

公開された仕様：この仕様（セクション8.3.2を参照）。

Applications that use this media type: N/A

このメディアタイプを使用するアプリケーション：N / A

Fragment identifier considerations: N/A

フラグメント識別子の考慮事項：なし

Additional information:

追加情報：

Deprecated alias names for this type: N/A

このタイプの非推奨のエイリアス名：N / A

```
Magic number(s):  N/A
```

```
File extension(s):  N/A
```

```
Macintosh file type code(s):  N/A
```

Person and email address to contact for further information: See Authors' Addresses section.

詳細について連絡する人とメールアドレス：作者のアドレスセクションを参照してください。

Intended usage: COMMON

使用目的：COMMON

Restrictions on usage: N/A

使用上の制限：N / A

Author: See Authors' Addresses section.

作成者：「作成者のアドレス」セクションを参照してください。

Change controller: IESG

コントローラーの変更：IESG

8.4. Transfer Coding Registry

The "HTTP Transfer Coding Registry" defines the namespace for transfer coding names. It is maintained at <http://www.iana.org/assignments/http-parameters>.

8.4. 転送コーディングレジストリ

「HTTP転送コーディングレジストリ」は、転送コーディング名の名前空間を定義します。<http://www.iana.org/assignments/http-parameters>で管理されています。

8.4.1. Procedure

Registrations MUST include the following fields:

o Name

o Description

o Pointer to specification text

Names of transfer codings MUST NOT overlap with names of content codings (Section 3.1.2.1 of [RFC7231]) unless the encoding transformation is identical, as is the case for the compression codings defined in Section 4.2.

Values to be added to this namespace require IETF Review (see Section 4.1 of [RFC5226]), and MUST conform to the purpose of transfer coding defined in this specification.

Use of program names for the identification of encoding formats is not desirable and is discouraged for future encodings.

8.4.1. 手順

登録には次のフィールドを含める必要があります。

o 名前

o 説明

o 仕様テキストへのポインタ

セクション4.2で定義されている圧縮コーディングの場合のように、エンコーディング変換が同一でない限り、転送コーディングの名前はコンテンツコーディングの名前（[RFC7231]のセクション3.1.2.1）と重複してはなりません（MUST NOT）。

この名前空間に追加される値は、IETFレビュー（[RFC5226]のセクション4.1を参照）を必要とし、この仕様で定義されている転送コーディングの目的に準拠する必要があります。

エンコーディング形式の識別にプログラム名を使用することは望ましくないため、将来のエンコーディングでは推奨されません。

8.4.2. Registration

The "HTTP Transfer Coding Registry" has been updated with the registrations below:

8.4.2. 登録

「HTTP Transfer Coding Registry」は、以下の登録で更新されています。

Name	Description	Reference
chunked	Transfer in a series of chunks	Section 4.1
compress	UNIX "compress" data format [Welch]	Section 4.2.1
deflate	"deflate" compressed data ([RFC1951]) inside the "zlib" data format ([RFC1950])	Section 4.2.2
gzip	GZIP file format [RFC1952]	Section 4.2.3
x-compress	Deprecated (alias for compress)	Section 4.2.1

```
| x-gzip     | Deprecated (alias for gzip)          | Section 4.2.3 |
+------------+--------------------------------------+---------------+
```

8.5. Content Coding Registration

IANA maintains the "HTTP Content Coding Registry" at <http://www.iana.org/assignments/http-parameters>.

The "HTTP Content Coding Registry" has been updated with the registrations below:

8.5. コンテンツコーディング登録

IANAは、<http://www.iana.org/assignments/http-parameters>で「HTTPコンテンツコーディングレジストリ」を維持しています。

「HTTPコンテンツコーディングレジストリ」は、以下の登録で更新されています。

Name	Description	Reference
compress	UNIX "compress" data format [Welch]	Section 4.2.1
deflate	"deflate" compressed data ([RFC1951]) inside the "zlib" data format ([RFC1950])	Section 4.2.2
gzip	GZIP file format [RFC1952]	Section 4.2.3
x-compress	Deprecated (alias for compress)	Section 4.2.1
x-gzip	Deprecated (alias for gzip)	Section 4.2.3

8.6. Upgrade Token Registry

The "Hypertext Transfer Protocol (HTTP) Upgrade Token Registry" defines the namespace for protocol-name tokens used to identify protocols in the Upgrade header field. The registry is maintained at <http://www.iana.org/assignments/http-upgrade-tokens>.

8.6. トークンレジストリのアップグレード

「ハイパーテキスト転送プロトコル（HTTP）アップグレードトークンレジストリ」は、アップグレードヘッダーフィールドでプロトコルを識別するために使用されるプロトコル名トークンのネームスペースを定義します。レジストリは<http://www.iana.org/assignments/http-upgrade-tokens>で管理されています。

8.6.1. Procedure

Each registered protocol name is associated with contact information and an optional set of specifications that details how the connection will be processed after it has been upgraded.

Registrations happen on a "First Come First Served" basis (see Section 4.1 of [RFC5226]) and are subject to the following rules:

1. A protocol-name token, once registered, stays registered forever.

2. The registration MUST name a responsible party for the registration.

3. The registration MUST name a point of contact.

4. The registration MAY name a set of specifications associated with that token. Such specifications need not be publicly available.

5. The registration SHOULD name a set of expected "protocol-version" tokens associated with that token at the time of registration.

6. The responsible party MAY change the registration at any time. The IANA will keep a record of all such changes, and make them available upon request.

7. The IESG MAY reassign responsibility for a protocol token. This will normally only be used in the case when a responsible party cannot be contacted.

This registration procedure for HTTP Upgrade Tokens replaces that previously defined in Section 7.2 of [RFC2817].

8.6.1. 手順

登録された各プロトコル名は、連絡先情報と、アップグレード後の接続の処理方法の詳細を示すオプションの仕様セットに関連付けられています。

登録は「先着順」で行われ（[RFC5226]のセクション4.1を参照）、次のルールが適用されます。

1. プロトコル名トークンは、いったん登録されると、永久に登録されたままになります。

2. 登録では、登録の責任者を指定する必要があります。

3. 登録では、連絡先を指定する必要があります。

4. 登録は、そのトークンに関連する一連の仕様を指定してもよい（MAY）。そのような仕様は公開されている必要はありません。

5. 登録は、登録時にそのトークンに関連付けられた、予想される「プロトコルバージョン」トークンのセットに名前を付ける必要があります（SHOULD）。

6. 責任者はいつでも登録を変更できます。 IANAはそのようなすべての変更の記録を保持し、要求に応じてそれらを利用可能にします。

7. IESGは、プロトコルトークンの責任を再割り当てしてもよい（MAY）。これは通常、責任者に連絡できない場合にのみ使用されます。

このHTTPアップグレードトークンの登録手順は、[RFC2817]のセクション7.2で以前に定義されたものに代わるものです。

8.6.2. Upgrade Token Registration

The "HTTP" entry in the upgrade token registry has been updated with the registration below:

8.6.2. トークン登録のアップグレード

アップグレードトークンレジストリの「HTTP」エントリは、以下の登録で更新されています。

Value	Description	Expected Version Tokens	Reference
HTTP	Hypertext Transfer Protocol	any DIGIT.DIGIT (e.g, "2.0")	Section 2.6

The responsible party is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

責任者は「IETF（iesg@ietf.org）-Internet Engineering Task Force」です。

9. Security Considerations

This section is meant to inform developers, information providers, and users of known security considerations relevant to HTTP message syntax, parsing, and routing. Security considerations about HTTP semantics and payloads are addressed in [RFC7231].

9. セキュリティに関する考慮事項

このセクションは、HTTPメッセージの構文、解析、およびルーティングに関連する既知のセキュリティの考慮事項について、開発者、情報プロバイダー、およびユーザーに通知することを目的としています。 HTTPセマンティクスとペイロードに関するセキュリティの考慮事項は、[RFC7231]で対処されています。

9.1. Establishing Authority

HTTP relies on the notion of an authoritative response: a response that has been determined by (or at the direction of) the authority identified within the target URI to be the most appropriate response for that request given the state of the target resource at the time of response message origination. Providing a response from a non-authoritative source, such as a shared cache, is often useful to improve performance and availability, but only to the extent that the source can be trusted or the distrusted response can be safely used.

9.1. 権限の確立

HTTPは、信頼できる応答の概念に依存しています。つまり、ターゲットURI内で（またはその方向で）識別された応答が、その時点でのターゲットリソースの状態から、その要求に最も適した応答であると判断されます。応答メッセージの発信。共有キャッシュなどの権限のないソースからの応答を提供することは、多くの場合、パフォーマンスと可用性を向上させるのに役立ちますが、ソースが信頼できるか、信頼できない応答を安全に使用できる場合に限られます。

Unfortunately, establishing authority can be difficult. For example, phishing is an attack on the user's perception of authority, where that perception can be misled by presenting similar branding in hypertext, possibly aided by userinfo obfuscating the authority component (see Section 2.7.1). User agents can reduce the impact of phishing attacks by enabling users to easily inspect a target URI prior to making an action, by prominently distinguishing (or rejecting) userinfo when present, and by not sending stored credentials and cookies when the referring document is from an unknown or untrusted source.

残念ながら、権限の確立は難しい場合があります。たとえば、フィッシングはユーザーの権限に対する認識への攻撃であり、ユーザー情報が権限コンポーネントを難読化することで、同様のブランディングをハイパーテキストで表示することにより、その認識を誤解させることができます（セクション2.7.1を参照）。ユーザーエージェントは、ユーザーがアクションを実行する前にターゲットURIを簡単に検査できるようにし、存在する場合はユーザー情報を目立つように区別（または拒否）し、参照ドキュメントがからのものである場合は保存された資格情報とCookieを送信しないことで、フィッシング攻撃の影響を軽減できます。不明または信頼できないソース。

When a registered name is used in the authority component, the "http" URI scheme (Section 2.7.1) relies on the user's local name resolution service to determine where it can find authoritative responses. This means that any attack on a user's network host table, cached names, or name resolution libraries becomes an avenue for attack on establishing authority. Likewise, the user's choice of server for Domain Name Service (DNS), and the hierarchy of servers from which it obtains resolution results, could impact the authenticity of address mappings; DNS Security Extensions (DNSSEC, [RFC4033]) are one way to improve authenticity.

登録された名前が機関コンポーネントで使用される場合、「http」URIスキーム（セクション2.7.1）は、ユーザーのローカル名前解決サービスに依存して、信頼できる応答を見つけることができる場所を決定します。これは、ユーザーのネットワークホストテーブル、キャッシュされた名前、または名前解決ライブラリに対する攻撃が、権限の確立に対する攻撃の手段になることを意味します。同様に、ユーザーがドメインネームサービス（DNS）用に選択したサーバーと、サーバーが解決結果を取得するサーバーの階層は、アドレスマッピングの信頼性に影響を与える可能性があります。 DNSセキュリティ拡張機能（DNSSEC、[RFC4033]）は、信頼性を向上させる1つの方法です。

Furthermore, after an IP address is obtained, establishing authority for an "http" URI is vulnerable to attacks on Internet Protocol routing.

さらに、IPアドレスが取得された後、 "http" URIの認証局を確立すると、インターネットプロトコルルーティングに対する攻撃に対して脆弱になります。

The "https" scheme (Section 2.7.2) is intended to prevent (or at least reveal) many of these potential attacks on establishing

「https」スキーム（セクション2.7.2）は、ネゴシエートされたTLS接続が保護されており、クライアントが通信サーバーのIDが

authority, provided that the negotiated TLS connection is secured and the client properly verifies that the communicating server's identity matches the target URI's authority component (see [RFC2818]). Correctly implementing such verification can be difficult (see [Georgiev]).

ターゲットURIの権限コンポーネント（[RFC2818]を参照）。そのような検証を正しく実装することは難しい場合があります（[Georgiev]を参照）。

9.2. Risks of Intermediaries

By their very nature, HTTP intermediaries are men-in-the-middle and, thus, represent an opportunity for man-in-the-middle attacks. Compromise of the systems on which the intermediaries run can result in serious security and privacy problems. Intermediaries might have access to security-related information, personal information about individual users and organizations, and proprietary information belonging to users and content providers. A compromised intermediary, or an intermediary implemented or configured without regard to security and privacy considerations, might be used in the commission of a wide range of potential attacks.

9.2. 仲介人のリスク

その性質上、HTTP仲介者は中間者であり、したがって中間者攻撃の機会を表します。仲介者が実行されるシステムの侵害は、深刻なセキュリティとプライバシーの問題を引き起こす可能性があります。仲介者は、セキュリティ関連の情報、個々のユーザーと組織に関する個人情報、およびユーザーとコンテンツプロバイダーに属する専有情報にアクセスできます。侵害された仲介者、またはセキュリティとプライバシーの考慮事項を考慮せずに実装または設定された仲介者が、広範な潜在的な攻撃のコミッションに使用される可能性があります。

Intermediaries that contain a shared cache are especially vulnerable to cache poisoning attacks, as described in Section 8 of [RFC7234].

[RFC7234]のセクション8で説明されているように、共有キャッシュを含む仲介者は、特にキャッシュポイズニング攻撃に対して脆弱です。

Implementers need to consider the privacy and security implications of their design and coding decisions, and of the configuration options they provide to operators (especially the default configuration).

実装者は、設計とコーディングの決定、およびオペレーターに提供する構成オプション（特にデフォルトの構成）のプライバシーとセキュリティへの影響を考慮する必要があります。

Users need to be aware that intermediaries are no more trustworthy than the people who run them; HTTP itself cannot solve this problem.

ユーザーは、仲介者がそれらを実行する人々よりも信頼できるものではないことを認識する必要があります。 HTTP自体はこの問題を解決できません。

9.3. Attacks via Protocol Element Length

Because HTTP uses mostly textual, character-delimited fields, parsers are often vulnerable to attacks based on sending very long (or very slow) streams of data, particularly where an implementation is expecting a protocol element with no predefined length.

9.3. プロトコル要素長による攻撃

HTTPは主にテキストで区切られたフィールドを使用するため、パーサーは、非常に長い（または非常に遅い）データストリームの送信に基づく攻撃に対して脆弱です。

To promote interoperability, specific recommendations are made for minimum size limits on request-line (Section 3.1.1) and header fields (Section 3.2). These are minimum recommendations, chosen to be supportable even by implementations with limited resources; it is expected that most implementations will choose substantially higher limits.

相互運用性を促進するために、リクエストライン（セクション3.1.1）およびヘッダーフィールド（セクション3.2）の最小サイズ制限について特定の推奨事項が作成されます。これらは最小限の推奨事項であり、リソースが限られている実装でもサポートできるように選択されています。ほとんどの実装では、大幅に高い制限が選択されると予想されます。

A server can reject a message that has a request-target that is too long (Section 6.5.12 of [RFC7231]) or a request payload that is too large (Section 6.5.11 of [RFC7231]). Additional status codes related to capacity limits have been defined by extensions to HTTP [RFC6585].

サーバーは、リクエストターゲットが長すぎるメッセージ（[RFC7231]のセクション6.5.12）または大きすぎるリクエストペイロード（[RFC7231]のセクション6.5.11）を含むメッセージを拒否できます。容量制限に関連する追加のステータスコードは、HTTP [RFC6585]の拡張機能によって定義されています。

Recipients ought to carefully limit the extent to which they process other protocol elements, including (but not limited to) request methods, response status phrases, header field-names, numeric values, and body chunks. Failure to limit such processing can result in buffer overflows, arithmetic overflows, or increased vulnerability to denial-of-service attacks.

受信者は、リクエストメソッド、応答ステータスフレーズ、ヘッダーフィールド名、数値、ボディチャンクを含む（ただしこれらに限定されない）他のプロトコル要素を処理する範囲を慎重に制限する必要があります。このような処理を制限しないと、バッファオーバーフロー、算術オーバーフロー、またはサービス拒否攻撃に対する脆弱性が高まる可能性があります。

9.4. Response Splitting

Response splitting (a.k.a, CRLF injection) is a common technique, used in various attacks on Web usage, that exploits the line-based nature of HTTP message framing and the

9.4. 応答分割

応答分割（別名、CRLFインジェクション）は、HTTPのメッセージフレーミングの行ベースの性質と、永続的な接続での応答への要求の順序付けされた関連付け[Klein]を利用する、Web使

ordered association of requests to responses on persistent connections [Klein]. This technique can be particularly damaging when the requests pass through a shared cache.

Response splitting exploits a vulnerability in servers (usually within an application server) where an attacker can send encoded data within some parameter of the request that is later decoded and echoed within any of the response header fields of the response. If the decoded data is crafted to look like the response has ended and a subsequent response has begun, the response has been split and the content within the apparent second response is controlled by the attacker. The attacker can then make any other request on the same persistent connection and trick the recipients (including intermediaries) into believing that the second half of the split is an authoritative answer to the second request.

For example, a parameter within the request-target might be read by an application server and reused within a redirect, resulting in the same parameter being echoed in the Location header field of the response. If the parameter is decoded by the application and not properly encoded when placed in the response field, the attacker can send encoded CRLF octets and other content that will make the application's single response look like two or more responses.

A common defense against response splitting is to filter requests for data that looks like encoded CR and LF (e.g., "%0D" and "%0A"). However, that assumes the application server is only performing URI decoding, rather than more obscure data transformations like charset transcoding, XML entity translation, base64 decoding, sprintf reformatting, etc. A more effective mitigation is to prevent anything other than the server's core protocol libraries from sending a CR or LF within the header section, which means restricting the output of header fields to APIs that filter for bad octets and not allowing application servers to write directly to the protocol stream.

9.5. Request Smuggling

Request smuggling ([Linhart]) is a technique that exploits differences in protocol parsing among various recipients to hide additional requests (which might otherwise be blocked or disabled by policy) within an apparently harmless request. Like response splitting, request smuggling can lead to a variety of attacks on HTTP usage.

This specification has introduced new requirements on request parsing, particularly with regard to message framing in Section 3.3.3, to reduce the effectiveness of request smuggling.

9.6. Message Integrity

HTTP does not define a specific mechanism for ensuring message integrity, instead relying on the error-detection ability of underlying transport protocols and the use of length or chunk-delimited framing to detect completeness. Additional integrity mechanisms, such as hash functions or digital signatures applied to the content, can be selectively added to messages via extensible metadata header fields. Historically, the lack of a single integrity mechanism has been justified by the informal nature of most HTTP communication. However, the

用に対するさまざまな攻撃で使用される一般的な手法です。この手法は、リクエストが共有キャッシュを通過するときに特に悪影響を与える可能性があります。

応答分割は、サーバー（通常はアプリケーションサーバー内）の脆弱性を悪用します。攻撃者は、要求の一部のパラメーター内でエンコードされたデータを送信し、後でデコードして、応答の応答ヘッダーフィールド内でエコーすることができます。デコードされたデータが、応答が終了し、その後の応答が始まったように見えるように細工されている場合、応答は分割されており、見かけ上の2番目の応答内のコンテンツは攻撃者によって制御されています。その後、攻撃者は同じ永続的な接続で他の要求を行い、受信者（仲介者を含む）をだまして、分割の後半が2番目の要求に対する信頼できる回答であると信じ込ませることができます。

たとえば、request-target内のパラメーターがアプリケーションサーバーによって読み取られ、リダイレクト内で再利用されると、同じパラメーターが応答のLocationヘッダーフィールドにエコーされます。パラメータがアプリケーションによってデコードされ、応答フィールドに配置されたときに適切にエンコードされていない場合、攻撃者はエンコードされたCRLFオクテットや、アプリケーションの単一の応答を2つ以上の応答のようにする他のコンテンツを送信できます。

応答分割に対する一般的な防御策は、エンコードされたCRとLF（たとえば、「%0D」と「%0A」）のように見えるデータのリクエストをフィルタリングすることです。ただし、これは、アプリケーションサーバーが文字セットのトランスコーディング、XMLエンティティの変換、base64のデコード、sprintfの再フォーマットなどのあいまいなデータ変換ではなく、URIのデコードのみを実行していることを前提としています。より効果的な緩和策は、サーバーのコアプロトコルライブラリ以外を防ぐことです。ヘッダーセクション内でCRまたはLFを送信しないこと。つまり、ヘッダーフィールドの出力を、不正なオクテットをフィルターするAPIに制限し、アプリケーションサーバーがプロトコルストリームに直接書き込むことを許可しないことを意味します。

9.5. 密輸を要請する

リクエストスマグリング（[Linhart]）は、さまざまな受信者間のプロトコル解析の違いを利用して、明らかに無害なリクエスト内に追加のリクエスト（ポリシーによってブロックまたは無効化される可能性がある）を隠す技術です。応答分割と同様に、リクエストスマグリングはHTTPの使用に対してさまざまな攻撃を引き起こす可能性があります。

この仕様では、特にセクション3.3.3のメッセージフレーミングに関して、リクエストの解析に関する新しい要件が導入され、リクエストの密輸の有効性が低減されています。

9.6. メッセージの整合性

HTTPはメッセージの整合性を確保するための特定のメカニズムを定義していません。代わりに、基礎となるトランスポートプロトコルのエラー検出機能と、完全性を検出するための長さまたはチャンク区切りのフレーミングの使用に依存しています。コンテンツに適用されるハッシュ関数やデジタル署名などの追加の整合性メカニズムは、拡張可能なメタデータヘッダーフィールドを介してメッセージに選択的に追加できます。歴史的に、単一の整合性メカニズムの欠如は、ほとんどのHTTP通信の

prevalence of HTTP as an information access mechanism has resulted in its increasing use within environments where verification of message integrity is crucial.

User agents are encouraged to implement configurable means for detecting and reporting failures of message integrity such that those means can be enabled within environments for which integrity is necessary. For example, a browser being used to view medical history or drug interaction information needs to indicate to the user when such information is detected by the protocol to be incomplete, expired, or corrupted during transfer. Such mechanisms might be selectively enabled via user agent extensions or the presence of message integrity metadata in a response. At a minimum, user agents ought to provide some indication that allows a user to distinguish between a complete and incomplete response message (Section 3.4) when such verification is desired.

9.7. Message Confidentiality

HTTP relies on underlying transport protocols to provide message confidentiality when that is desired. HTTP has been specifically designed to be independent of the transport protocol, such that it can be used over many different forms of encrypted connection, with the selection of such transports being identified by the choice of URI scheme or within user agent configuration.

The "https" scheme can be used to identify resources that require a confidential connection, as described in Section 2.7.2.

9.8. Privacy of Server Log Information

A server is in the position to save personal data about a user's requests over time, which might identify their reading patterns or subjects of interest. In particular, log information gathered at an intermediary often contains a history of user agent interaction, across a multitude of sites, that can be traced to individual users.

HTTP log information is confidential in nature; its handling is often constrained by laws and regulations. Log information needs to be securely stored and appropriate guidelines followed for its analysis. Anonymization of personal information within individual entries helps, but it is generally not sufficient to prevent real log traces from being re-identified based on correlation with other access characteristics. As such, access traces that are keyed to a specific client are unsafe to publish even if the key is pseudonymous.

To minimize the risk of theft or accidental publication, log information ought to be purged of personally identifiable information, including user identifiers, IP addresses, and user-provided query parameters, as soon as that information is no longer necessary to support operational needs for security, auditing, or fraud control.

10. Acknowledgments

This edition of HTTP/1.1 builds on the many contributions that went into RFC 1945, RFC 2068, RFC 2145, and RFC 2616, including substantial contributions made by the previous authors, editors, and Working Group Chairs: Tim Berners-Lee, Ari Luotonen, Roy T. Fielding, Henrik Frystyk Nielsen, Jim Gettys, Jeffrey C. Mogul, Larry Masinter, and Paul J. Leach. Mark Nottingham oversaw this effort as Working Group Chair.

非公式な性質によって正当化されてきました。ただし、情報アクセスメカニズムとしてのHTTPの普及により、メッセージの整合性の検証が重要な環境での使用が増加しています。

ユーザーエージェントは、メッセージの整合性の障害を検出および報告する構成可能な手段を実装して、整合性が必要な環境内でそれらの手段を有効にできるようにすることをお勧めします。たとえば、病歴または薬物相互作用情報を表示するために使用されているブラウザは、転送中にそのような情報がプロトコルによって検出されたときに、不完全、期限切れ、または破損していることをユーザーに示す必要があります。このようなメカニズムは、ユーザーエージェント拡張機能や応答のメッセージ整合性メタデータの存在を介して選択的に有効にすることができます。少なくとも、ユーザーエージェントは、そのような検証が必要な場合に、ユーザーが完全な応答メッセージと不完全な応答メッセージ（セクション3.4）を区別できるようにする何らかの指示を提供する必要があります。

9.7. メッセージの機密性

HTTPは、必要なときにメッセージの機密性を提供するために、基礎となるトランスポートプロトコルに依存しています。 HTTPはトランスポートプロトコルに依存しないように特別に設計されており、暗号化された接続のさまざまな形式で使用でき、そのようなトランスポートの選択はURIスキームの選択またはユーザーエージェント構成内で識別されます。

セクション2.7.2で説明されているように、「https」スキームを使用して、機密接続を必要とするリソースを特定できます。

9.8. サーバーログ情報のプライバシー

サーバーは、時間の経過に伴うユーザーのリクエストに関する個人データを保存する立場にあります。これにより、ユーザーの読み取りパターンや関心のある対象を特定できます。特に、仲介業者で収集されたログ情報には、多くのサイトにわたるユーザーエージェントの相互作用の履歴が含まれていることが多く、個々のユーザーまで追跡できます。

HTTPログ情報は本質的に機密情報です。その取り扱いは、多くの場合、法律や規制によって制約されています。ログ情報は安全に保管し、適切なガイドラインに従って分析する必要があります。個々のエントリ内の個人情報の匿名化は役立ちますが、通常、実際のログトレースが他のアクセス特性との相関に基づいて再識別されるのを防ぐには不十分です。そのため、特定のクライアントをキーとするアクセストレースは、キーが仮名であっても公開するのは安全ではありません。

盗難や偶発的な公開のリスクを最小限に抑えるために、ユーザー識別子、IPアドレス、ユーザー指定のクエリパラメーターなどの個人を特定できる情報は、セキュリティの運用ニーズをサポートするために必要でなくなったらすぐに、ログ情報から削除する必要があります。 、監査、または詐欺対策。

10. 謝辞

この版のHTTP / 1.1は、RFC 1945、RFC 2068、RFC 2145、およびRFC 2616に組み込まれた多くの貢献に基づいて構築されています。これには、以前の著者、編集者、およびワーキンググループの議長による貢献も含まれます。TimBerners-Lee、Ari Luotonen 、ロイ・T・フィールディング、ヘンリック・フリスティク・ニールセン、ジム・ゲティス、ジェフリー・C・モーグ

ル、ラリー・マシンター、ポール・J・リーチ。マークノッティンガムは、この取り組みをワーキンググループチェアとして監督しました。

Since 1999, the following contributors have helped improve the HTTP specification by reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues:

1999年以降、次の貢献者がバグの報告、賢明な質問の実施、草稿の作成またはレビュー、および未解決の問題の評価を行うことにより、HTTP仕様の改善に貢献しています。

Adam Barth, Adam Roach, Addison Phillips, Adrian Chadd, Adrian Cole, Adrien W. de Croy, Alan Ford, Alan Ruttenberg, Albert Lunde, Alek Storm, Alex Rousskov, Alexandre Morgaut, Alexey Melnikov, Alisha Smith, Amichai Rothman, Amit Klein, Amos Jeffries, Andreas Maier, Andreas Petersson, Andrei Popov, Anil Sharma, Anne van Kesteren, Anthony Bryan, Asbjorn Ulsberg, Ashok Kumar, Balachander Krishnamurthy, Barry Leiba, Ben Laurie, Benjamin Carlyle, Benjamin Niven-Jenkins, Benoit Claise, Bil Corry, Bill Burke, Bjoern Hoehrmann, Bob Scheifler, Boris Zbarsky, Brett Slatkin, Brian Kell, Brian McBarron, Brian Pane, Brian Raymor, Brian Smith, Bruce Perens, Bryce Nesbitt, Cameron Heavon-Jones, Carl Kugler, Carsten Bormann, Charles Fry, Chris Burdess, Chris Newman, Christian Huitema, Cyrus Daboo, Dale Robert Anderson, Dan Wing, Dan Winship, Daniel Stenberg, Darrel Miller, Dave Cridland, Dave Crocker, Dave Kristol, Dave Thaler, David Booth, David Singer, David W. Morris, Diwakar Shetty, Dmitry Kurochkin, Drummond Reed, Duane Wessels, Edward Lee, Eitan Adler, Eliot Lear, Emile Stephan, Eran Hammer-Lahav, Eric D. Williams, Eric J. Bowman, Eric Lawrence, Eric Rescorla, Erik Aronesty, EungJun Yi, Evan Prodromou, Felix Geisendoerfer, Florian Weimer, Frank Ellermann, Fred Akalin, Fred Bohle, Frederic Kayser, Gabor Molnar, Gabriel Montenegro, Geoffrey Sneddon, Gervase Markham, Gili Tzabari, Grahame Grieve, Greg Slepak, Greg Wilkins, Grzegorz Calkowski, Harald Tveit Alvestrand, Harry Halpin, Helge Hess, Henrik Nordstrom, Henry S. Thompson, Henry Story, Herbert van de Sompel, Herve Ruellan, Howard Melman, Hugo Haas, Ian Fette, Ian Hickson, Ido Safruti, Ilari Liusvaara, Ilya Grigorik, Ingo Struck, J. Ross Nicoll, James Cloos, James H. Manger, James Lacey, James M. Snell, Jamie Lokier, Jan Algermissen, Jari Arkko, Jeff Hodges (who came up with the term 'effective Request-URI'), Jeff Pinner, Jeff Walden, Jim Luther, Jitu Padhye, Joe D. Williams, Joe Gregorio, Joe Orton, Joel Jaeggli, John C. Klensin, John C. Mallery, John Cowan, John Kemp, John Panzer, John Schneider, John Stracke, John Sullivan, Jonas Sicking, Jonathan A. Rees, Jonathan Billington, Jonathan Moore, Jonathan Silvera, Jordi Ros, Joris Dobbelsteen, Josh Cohen, Julien Pierre, Jungshik Shin, Justin Chapweske, Justin Erenkrantz, Justin James, Kalvinder Singh, Karl Dubost, Kathleen Moriarty, Keith Hoffman, Keith Moore, Ken Murchison, Koen Holtman, Konstantin Voronkov, Kris Zyp, Leif Hedstrom, Lionel Morand, Lisa Dusseault, Maciej Stachowiak, Manu Sporny, Marc Schneider, Marc Slemko, Mark Baker, Mark Pauley, Mark Watson, Markus Isomaki, Markus Lanthaler, Martin J. Duerst, Martin Musatov, Martin Nilsson, Martin Thomson, Matt Lynch, Matthew Cox, Matthew Kerwin, Max Clark, Menachem Dodge, Meral Shirazipour, Michael Burrows, Michael Hausenblas, Michael Scharf, Michael Sweet, Michael Tuexen, Michael Welzl, Mike Amundsen, Mike Belshe, Mike Bishop, Mike Kelly, Mike Schinkel, Miles Sabin, Murray S. Kucherawy, Mykyta Yevstifeyev, Nathan Rixham, Nicholas Shanks, Nico Williams, Nicolas Alvarez, Nicolas Mailhot, Noah Slater, Osama Mazahir, Pablo Castro, Pat Hayes, Patrick R. McManus, Paul E. Jones, Paul Hoffman, Paul Marquess, Pete Resnick, Peter Lepeska,

アダム・バース、アダム・ローチ、アディソン・フィリップス、エイドリアン・チャッド、エイドリアン・コール、エイドリアン・W・デ・クロイ、アラン・フォード、アラン・ルッテンバーグ、アルバート・ランデ、アレク・ストーム、アレックス・ルスコフ、アレクサンドル・モルゴー、アレクセイ・メルニコフ、アリシャ・スミス、アミチャイ・ロスマン、アミット・クライン、Amos Jeffries、Andreas Maier、Andreas Petersson、Andrei Popov、Anil Sharma、Anne van Kesteren、Anthony Bryan、Asbjorn Ulsberg、Ashok Kumar、Balachander Krishnamurthy、Barry Leiba、Ben Laurie、Benjamin Carlyle、Benjamin Niven-Jenkins、Benoコリー、ビルバーク、ビョーンホーマン、ボブシャイフラー、ボリスズバースキー、ブレットスラットキン、ブライアンケル、ブライアンマクバロン、ブライアンペイン、ブライアンレイモア、ブライアンスミス、ブルースペレンズ、ブライスネスビット、キャメロンヘボンジョーンズ、カールクグラー、カーステンボーマン、チャールズFry、Chris Burdess、Chris Newman、Christian Huitema、Cyrus Daboo、Dale Robert Anderson、Dan Wing、Dan Winship、Daniel Stenberg、Darrel Miller、Dave Cridland、Dave Crocker、Dave Kristol、Dave Thaler、David Booth、David Singer、David W 。Morris、Diwakar Shetty、Dmitry Kurochkイン、ドラモンドリード、デュアンウェッセル、エドワードリー、エイタンアドラー、エリオットリア、エミールステファン、エランハマーラハブ、エリックD.ウィリアムズ、エリックJ.ボーマン、エリックローレンス、エリックレスコーラ、エリックアロネスティ、ウンジュンイ、エヴァンプロドロムー、 Felix Geisendoerfer、Florian Weimer、Frank Ellermann、Fred Akalin、Fred Bohle、Frederic Kayser、Gabor Molnar、Gabriel Montenegro、Geoffrey Sneddon、Gervase Markham、Gili Tzabari、Grahame Grieve、Greg Slepak、Greg Wilkins、Grzegorvest Carzegorz Char Halpin、Helge Hess、Henrik Nordstrom、Henry S.Thompson、Henry Story、Herbert van de Sompel、Herve Ruellan、Howard Melman、Hugo Haas、Ian Fette、Ian Hickson、Ido Safruti、Ilari Liusvaara、Ilya Grigorik、Ingo Struck、J。ロスニコル、ジェームズクロース、ジェームスH.マンガー、ジェームスレイシー、ジェームズM.スネル、ジェイミーロキエ、ヤンアルガーミセン、ジャリアルコ、ジェフホッジス（「効果的なリクエストURI」という用語を思いついた人）、ジェフピナー、ジェフウォルデン、Jim Luther、Jitu Padhye、Joe D. Williams、Joe Gregorio、Joe Orton、Joel Jaeggli、John C.K lensin、ジョン・C・マレリー、ジョン・コーワン、ジョン・ケンプ、ジョン・パンツァー、ジョン・シュナイダー、ジョン・ストラック、ジョン・サリバン、ジョナス・シッキング、ジョナサン・A・リース、ジョナサン・ビリントン、ジョナサン・ムーア、ジョナサン・シルラ、ジョルディ・ロス、ジョリス・ドベルスティーン、ジョシュ・コーエン、ジュリアン・ピエール、ユングシク・シン、ジャスティン・チャプウェスケ、ジャスティン・エレンクランツ、ジャスティン・ジェームス、カルビンダー・シン、カール・デュボスト、キャスリーン・モリアーティ、キース・ホフマン、キース・ムーア、ケン・マーチソン、ケン・ホルトマン、コンスタンティン・ボロンコフ、クリス・ジプ、レイフ・ヘドストローム・ライオン Dusseault、Maciej Stachowiak、Manu Sporny、Marc Schneider、Marc Slemko、Mark Baker、Mark Pauley、Mark Watson、Markus Isomaki、Markus Lanthaler、Martin J.

Peter Occil, Peter Saint-Andre, Peter Watkins, Phil Archer, Phil Hunt, Philippe Mougin, Phillip Hallam-Baker, Piotr Dobrogost, Poul-Henning Kamp, Preethi Natarajan, Rajeev Bector, Ray Polk, Reto Bachmann-Gmuer, Richard Barnes, Richard Cyganiak, Rob Trace, Robby Simpson, Robert Brewer, Robert Collins, Robert Mattson, Robert O'Callahan, Robert Olofsson, Robert Sayre, Robert Siemer, Robert de Wilde, Roberto Javier Godoy, Roberto Peon, Roland Zink, Ronny Widjaja, Ryan Hamilton, S. Mike Dierken, Salvatore Loreto, Sam Johnston, Sam Pullara, Sam Ruby, Saurabh Kulkarni, Scott Lawrence (who maintained the original issues list), Sean B. Palmer, Sean Turner, Sebastien Barnoud, Shane McCarron, Shigeki Ohtsu, Simon Yarde, Stefan Eissing, Stefan Tilkov, Stefanos Harhalakis, Stephane Bortzmeyer, Stephen Farrell, Stephen Kent, Stephen Ludin, Stuart Williams, Subbu Allamaraju, Subramanian Moonesamy, Susan Hares, Sylvain Hellegouarch, Tapan Divekar, Tatsuhiro Tsujikawa, Tatsuya Hayashi, Ted Hardie, Ted Lemon, Thomas Broyer, Thomas Fossati, Thomas Maslen, Thomas Nadeau, Thomas Nordin, Thomas Roessler, Tim Bray, Tim Morgan, Tim Olsen, Tom Zhou, Travis Snoozy, Tyler Close, Vincent Murphy, Wenbo Zhu, Werner Baumann, Wilbur Streett, Wilfredo Sanchez Vega, William A. Rowe Jr., William Chan, Willy Tarreau, Xiaoshu Wang, Yaron Goland, Yngve Nysaeter Pettersen, Yoav Nir, Yogesh Bang, Yuchung Cheng, Yutaka Oiwa, Yves Lafon (long-time member of the editor team), Zed A. Shaw, and Zhong Yu.

See Section 16 of [RFC2616] for additional acknowledgements from prior revisions.

Duerst、Martin Musatov、Martin Nilsson、Martin Thomson、Matt Lynch、Matthew Cox、Matthewカーウィン、マックスクラーク、メナケムダッジ、メラルシラジプール、マイケルバロウズ、マイケルハウゼンブラス、マイケルシャーフ、マイケルスウィート、マイケルウェルセン、マイクアムンセン、マイクベルシェ、マイクビショップ、マイクケリー、マイクシンケル、マイルズセービン、マレーS.クチェラウィ、ミキタ・エフスティfeyev、Nathan Rixham、Nicholas Shanks、Nico Williams、Nicolas Alvarez、Nicolas Mailhot、Noah Slater、Osama Mazahir、Pablo Castro、Pat Hayes、Patrick R. McManus、Paul E. Jones、Paul Hoffman、Paul Marquess、Pete Resnick、Peter Lepeska 、ピーターオクシル、ピーターセントアンドレ、ピーターワトキンス、フィルアーチャー、フィルハント、フィリップムジャン、フィリップハラムベイカー、ピョートルドブロゴスト、ポールヘニングカンプ、プリティナタラジャン、ラジーブベクター、レイポーク、レトバックマングミュアー、リチャードバーンズ、リチャードシガニアック、ロブトレース、ロビーシンプソン、ロバートブリューワー、ロバートコリンズ、ロバートマットソン、ロバートオキャラハン、ロバートオロフソン、ロバートセイヤー、ロバートシーマー、ロバートドワイルド、ロベルトハビエルゴドイ、ロベルトペオン、ロランドジンク、ロニーウィジャヤ、ライアンハミルトン、S。マイクディアケン、サルバトーレロレート、サムジョンストン、サムプルララ、サムルビー、サウラブカルカルニ、スコットローレンス（元の問題リストを維持）、ショーンB.パーマー、ショーンターナー、セバスチャンバーヌード、シェーンマッカロン、大津茂樹、Simon Yarde、Stefan Eissing、Stefan Tilkov、Stefanos Harhalakis、Stephane Bortzmeyer、Stephen Farrell、Stephen Kent、Stephen Ludin、Stuart Williams、Subbu Allamaraju、Subramanian Moonesamy、Susan Hares、Sylvain Hellegouarch、Tapan Divekar、Tatsuhiro Tsujikawa、Tatsuya Hayashi、Ted Hardie、Ted Lemon、Thomas Broyer Thomas Maslen、Thomas Nadeau、Thomas Nordin、Thomas Roessler、Tim Bray、Tim Morgan、Tim Olsen、Tom Zhou、Travis Snoozy、Tyler Close、Vincent Murphy、Wenbo Zhu、Werner Baumann、Wilbur Streett、Wilfredo Sanchez Vega、William A. Rowe Jr.、William Chan、Willy Tarreau、Xiaoshu Wang、Yaron Goland、Yngve Nysaeter Pettersen、Yoav Nir、Yogesh Bang、Yuchung Cheng、Yutaka Oiwa、Yves Lafon（編集チームの長いメンバー）、Zed A. Shaw、そして中玉。

以前の改訂からの追加の謝辞については、[RFC2616]のセクション16を参照してください。

11. References

11.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1950] Deutsch, L. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66,

11. 参考文献

11.1. 引用文献

[RFC0793] Postel、J。、「Transmission Control Protocol」、STD 7、RFC 793、1981年9月。

[RFC1950] Deutsch、L. and J-L。ゲイリー、「ZLIB圧縮データ形式仕様バージョン3.3」、RFC 1950、1996年5月。

[RFC1951] Deutsch、P。、「DEFLATE Compressed Data Format Specification version 1.3」、RFC 1951、1996年5月。

[RFC1952] Deutsch、P.、Gailly、J-L。、Adler、M.、Deutsch、L。、およびG. Randers-Pehrson、「GZIPファイル形式仕様バージョン4.3」、RFC 1952、1996年5月。

[RFC2119] Bradner、S。、「要件レベルを示すためにRFCで使用するキーワード」、BCP 14、RFC 2119、1997年3月。

[RFC3986] Berners-Lee、T.、Fielding、R。、およびL. Masinter、「Uniform Resource Identifier（URI）：Generic

RFC 3986, January 2005.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, June 2014.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

[Welch] Welch, T., "A Technique for High-Performance Data Compression", IEEE Computer 17(6), June 1984.

11.2. Informative References

[BCP115] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", BCP 115, RFC 4395, February 2006.

[BCP13] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.

[BCP90] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[Georgiev] Georgiev, M., Iyengar, S., Jana, S., Anubhai, R., Boneh, D., and V. Shmatikov, "The Most Dangerous Code in the World: Validating SSL Certificates in Non-browser Software", In Proceedings of the 2012 ACM Conference on Computer and Communications Security (CCS '12), pp. 38-49, October 2012, <http://doi.acm.org/10.1145/2382196.2382204>.

[ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.

[Klein] Klein, A., "Divide and Conquer - HTTP Response Splitting, Web Cache Poisoning Attacks, and Related Topics", March 2004, <http://packetstormsecurity.com/papers/general/whitepaper_httpresponse.pdf>.

[Kri2001] Kristol, D., "HTTP Cookies: Standards, Privacy, and Politics", ACM Transactions on Internet Technology 1(2), November 2001, <http://arxiv.org/abs/cs.SE/0105018>.

RFC 3986、2005年1月。

[RFC5234]クロッカー、D。、エド。およびP. Overell、「構文仕様の拡張BNF：ABNF」、STD 68、RFC 5234、2008年1月。

[RFC7231]フィールディング、R。、エド。およびJ. Reschke編、「Hypertext Transfer Protocol（HTTP / 1.1）：Semantics and Content」、RFC 7231、2014年6月。

[RFC7232]フィールディング、R。、エド。およびJ. Reschke編、「Hypertext Transfer Protocol（HTTP / 1.1）：Conditional Requests」、RFC 7232、2014年6月。

[RFC7233] Fielding、R。、編、Lafon、Y。、編、およびJ. Reschke、編、「Hypertext Transfer Protocol（HTTP / 1.1）：Range Requests」、RFC 7233、2014年6月。

[RFC7234] Fielding、R.、Ed。、Nottingham、M.、Ed。、and J. Reschke、Ed。、 "Hypertext Transfer Protocol（HTTP / 1.1）：Caching"、RFC 7234、June 2014。

[RFC7235]フィールディング、R。、エド。およびJ. Reschke編、「Hypertext Transfer Protocol（HTTP / 1.1）：Authentication」、RFC 7235、2014年6月。

[USASCII] American National Standards Institute、「Coded Character Set-7-bit American Standard Code for Information Interchange」、ANSI X3.4、1986。

[ウェルチ]ウェルチ、T。、「高性能データ圧縮の技法」、IEEE Computer 17（6）、1984年6月。

11.2. 参考引用

[BCP115] Hansen、T.、Hardie、T。、およびL. Masinter、「新しいURIスキームのガイドラインと登録手順」、BCP 115、RFC 4395、2006年2月。

[BCP13] Freed、N.、Klensin、J。、およびT. Hansen、「メディアタイプの仕様と登録手順」、BCP 13、RFC 6838、2013年1月。

[BCP90] Klyne、G.、Nottingham、M。、およびJ. Mogul、「メッセージヘッダーフィールドの登録手順」、BCP 90、RFC 3864、2004年9月。

[Georgiev] Georgiev、M.、Iyengar、S.、Jana、S.、Anubhai、R.、Boneh、D。、およびV. Shmatikov、「世界で最も危険なコード：非ブラウザソフトウェアでのSSL証明書の検証"、コンピュータと通信のセキュリティに関する2012年ACM会議の議事録（CCS '12）、38-49ページ、2012年10月、<http://doi.acm.org/10.1145/2382196.2382204>。

[ISO-8859-1]国際標準化機構、「情報技術-8ビットシングルバイトコード化グラフィック文字セット-パート1：ラテンアルファベットNo. 1」、ISO / IEC 8859-1：1998、1998。

[Klein] Klein、A。、「Divide and Conquer-HTTP Response Splitting、Web Cache Poisoning Attacks、and Related Topics」、2004年3月、<http://packetstormsecurity.com/papers/general/whitepaper_httpresponse.pdf>。

[Kri2001] Kristol、D。、「HTTP Cookies：Standards、Privacy、and Politics」、ACM Transactions on Internet Technology 1（2）、2001年11月、

<http://arxiv.org/abs/cs.SE/0105018> 。

[Linhart] Linhart, C., Klein, A., Heled, R., and S. Orrin, "HTTP Request Smuggling", June 2005, <http://www.watchfire.com/news/whitepapers.aspx>.

[Linhart] Linhart、C.、Klein、A.、Heled、R。、およびS. Orrin、「HTTP Request Smuggling」、2005年6月、<http://www.watchfire.com/news/whitepapers.aspx>。

[RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, March 1996.

[RFC1919] Chatel、M。、「Classical vs Transparent IP Proxies」、RFC 1919、1996年3月。

[RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

[RFC1945] Berners-Lee、T.、Fielding、R。、およびH. Nielsen、「Hypertext Transfer Protocol-HTTP / 1.0」、RFC 1945、1996年5月。

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2045] Freed、N。およびN. Borenstein、「Multipurpose Internet Mail Extensions（MIME）Part One：Format of Internet Message Bodies」、RFC 2045、1996年11月。

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[RFC2047]ムーアK。、「MIME（多目的インターネットメール拡張）パート3：非ASCIIテキストのメッセージヘッダー拡張」、RFC 2047、1996年11月。

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[RFC2068] Fielding、R.、Gettys、J.、Mogul、J.、Nielsen、H。、およびT. Berners-Lee、「Hypertext Transfer Protocol-HTTP / 1.1」、RFC 2068、1997年1月。

[RFC2145] Mogul, J., Fielding, R., Gettys, J., and H. Nielsen, "Use and Interpretation of HTTP Version Numbers", RFC 2145, May 1997.

[RFC2145] Mogul、J.、Fielding、R.、Gettys、J。、およびH. Nielsen、「HTTPバージョン番号の使用と解釈」、RFC 2145、1997年5月。

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2616] Fielding、R.、Gettys、J.、Mogul、J.、Frystyk、H.、Masinter、L.、Leach、P。、およびT. Berners-Lee、「ハイパーテキスト転送プロトコル-HTTP / 1.1」 、RFC 2616、1999年6月。

[RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.

[RFC2817] Khare、R。およびS. Lawrence、「HTTP / 1.1内のTLSへのアップグレード」、RFC 2817、2000年5月。

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2818] Rescorla、E。、「HTTP over TLS」、RFC 2818、2000年5月。

[RFC3040] Cooper, I., Melve, I., and G. Tomlinson, "Internet Web Replication and Caching Taxonomy", RFC 3040, January 2001.

[RFC3040]クーパー、I。、メルブ、I。、およびG.トムリンソン、「インターネットWebレプリケーションおよびキャッシング分類法」、RFC 3040、2001年1月。

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4033] Arends、R.、Austein、R.、Larson、M.、Massey、D。、およびS. Rose、「DNSセキュリティの概要と要件」、RFC 4033、2005年3月。

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006.

[RFC4559] Jaganathan、K.、Zhu、L。、およびJ. Brezak、「Microsoft WindowsでのSPNEGOベースのKerberosおよびNTLM HTTP認証」、RFC 4559、2006年6月。

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5226] Narten、T。およびH. Alvestrand、「RFCでIANAの考慮事項セクションを作成するためのガイドライン」、BCP 26、RFC 5226、2008年5月。

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5246] Dierks、T。およびE. Rescorla、「The Transport Layer Security（TLS）Protocol Version 1.2」、RFC 5246、2008年8月。

[RFC5322] Resnick, P., "Internet Message Format", RFC 5322, October 2008.

[RFC5322] Resnick、P。、「インターネットメッセージ形式」、RFC 5322、2008年10月。

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

[RFC6265]バース、A。、「HTTP状態管理メカニズム」、RFC 6265、2011年4月。

[RFC6585] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, April 2012.

[RFC6585]ノッティンガム、M。およびR.フィールディング、「追加のHTTPステータスコード」、RFC 6585、2012年4月。

Appendix A. HTTP Version History

HTTP has been in use since 1990. The first version, later referred to as HTTP/0.9, was a simple protocol for hypertext data transfer across the Internet, using only a single request method (GET) and no metadata. HTTP/1.0, as defined by [RFC1945], added a range of request methods and MIME-like messaging, allowing for metadata to be transferred and modifiers placed on the request/response semantics. However, HTTP/1.0 did not sufficiently take into consideration the effects of hierarchical proxies, caching, the need for persistent connections, or name-based virtual hosts. The proliferation of incompletely implemented applications calling themselves "HTTP/1.0" further necessitated a protocol version change in order for two communicating applications to determine each other's true capabilities.

HTTP/1.1 remains compatible with HTTP/1.0 by including more stringent requirements that enable reliable implementations, adding only those features that can either be safely ignored by an HTTP/1.0 recipient or only be sent when communicating with a party advertising conformance with HTTP/1.1.

HTTP/1.1 has been designed to make supporting previous versions easy. A general-purpose HTTP/1.1 server ought to be able to understand any valid request in the format of HTTP/1.0, responding appropriately with an HTTP/1.1 message that only uses features understood (or safely ignored) by HTTP/1.0 clients. Likewise, an HTTP/1.1 client can be expected to understand any valid HTTP/1.0 response.

Since HTTP/0.9 did not support header fields in a request, there is no mechanism for it to support name-based virtual hosts (selection of resource by inspection of the Host header field). Any server that implements name-based virtual hosts ought to disable support for HTTP/0.9. Most requests that appear to be HTTP/0.9 are, in fact, badly constructed HTTP/1.x requests caused by a client failing to properly encode the request-target.

付録A. HTTPバージョン履歴

HTTPは1990年から使用されています。最初のバージョン（後でHTTP / 0.9と呼ばれる）は、単一の要求メソッド（GET）のみを使用し、メタデータを使用しない、インターネット経由のハイパーテキストデータ転送用のシンプルなプロトコルでした。[RFC1945]で定義されているHTTP / 1.0は、一連のリクエストメソッドとMIMEのようなメッセージングを追加し、メタデータを転送し、修飾子をリクエスト/レスポンスセマンティクスに配置できるようにしました。ただし、HTTP / 1.0では、階層プロキシ、キャッシュ、永続的な接続の必要性、または名前ベースの仮想ホストの影響を十分に考慮していませんでした。 「HTTP / 1.0」と呼ばれる不完全に実装されたアプリケーションの急増により、通信している2つのアプリケーションが互いの真の機能を判別するために、プロトコルバージョンの変更がさらに必要になりました。

HTTP / 1.1は、HTTP / 1.0との互換性を維持し、信頼性の高い実装を可能にするより厳しい要件を組み込んで、HTTP / 1.0受信者が安全に無視できる機能、またはHTTP / 1.1に準拠していると宣伝しているパーティーと通信するときにのみ送信できる機能のみを追加します。 。

HTTP / 1.1は、以前のバージョンを簡単にサポートできるように設計されています。汎用HTTP / 1.1サーバーは、HTTP / 1.0の形式で有効な要求を理解でき、HTTP / 1.0クライアントが理解（または安全に無視）した機能のみを使用するHTTP / 1.1メッセージで適切に応答する必要があります。同様に、HTTP / 1.1クライアントは、有効なHTTP / 1.0応答を理解することが期待できます。

HTTP / 0.9はリクエストのヘッダーフィールドをサポートしていなかったため、名前ベースの仮想ホスト（ホストヘッダーフィールドの検査によるリソースの選択）をサポートするメカニズムはありません。名前ベースの仮想ホストを実装するサーバーは、HTTP / 0.9のサポートを無効にする必要があります。 HTTP / 0.9のように見えるほとんどの要求は、実際には、クライアントが要求ターゲットを適切にエンコードできないことによって引き起こされた、正しく構成されていないHTTP / 1.x要求です。

A.1. Changes from HTTP/1.0

This section summarizes major differences between versions HTTP/1.0 and HTTP/1.1.

A.1. HTTP / 1.0からの変更点

このセクションでは、バージョンHTTP / 1.0とHTTP / 1.1の主な違いを要約します。

A.1.1. Multihomed Web Servers

The requirements that clients and servers support the Host header field (Section 5.4), report an error if it is missing from an HTTP/1.1 request, and accept absolute URIs (Section 5.3) are among the most important changes defined by HTTP/1.1.

Older HTTP/1.0 clients assumed a one-to-one relationship of IP addresses and servers; there was no other established mechanism for distinguishing the intended server of a request than the IP address to which that request was directed. The Host header field was introduced during the development of HTTP/1.1 and, though it was quickly implemented by most HTTP/1.0 browsers, additional requirements were placed on all HTTP/1.1 requests in order to ensure complete adoption. At the time of this writing, most HTTP-based services are dependent upon the Host header field for targeting requests.

A.1.1. マルチホームWebサーバー

クライアントとサーバーがホストヘッダーフィールドをサポートする要件（セクション5.4）、HTTP / 1.1リクエストから欠落している場合にエラーを報告すること、および絶対URIを受け入れること（セクション5.3）は、HTTP / 1.1で定義された最も重要な変更の1つです。

古いHTTP / 1.0クライアントは、IPアドレスとサーバーの1対1の関係を想定していました。要求の対象となるサーバーを区別するための確立されたメカニズムは、その要求の送信先のIPアドレス以外にありませんでした。ホストヘッダーフィールドはHTTP / 1.1の開発中に導入され、ほとんどのHTTP / 1.0ブラウザーによってすぐに実装されましたが、完全な採用を確実にするために、すべてのHTTP / 1.1要求に追加の要件が課されました。この記事の執筆時点では、ほとんどのHTTPベースのサービスは、要求をターゲットにするためのHostヘッダーフィールドに依存しています。

A.1.2. Keep-Alive Connections

In HTTP/1.0, each connection is established by the client prior

A.1.2. キープアライブ接続

HTTP / 1.0では、各接続は要求の前にクライアントによって確立

to the request and closed by the server after sending the response. However, some implementations implement the explicitly negotiated ("Keep-Alive") version of persistent connections described in Section 19.7.1 of [RFC2068].

Some clients and servers might wish to be compatible with these previous approaches to persistent connections, by explicitly negotiating for them with a "Connection: keep-alive" request header field. However, some experimental implementations of HTTP/1.0 persistent connections are faulty; for example, if an HTTP/1.0 proxy server doesn't understand Connection, it will erroneously forward that header field to the next inbound server, which would result in a hung connection.

One attempted solution was the introduction of a Proxy-Connection header field, targeted specifically at proxies. In practice, this was also unworkable, because proxies are often deployed in multiple layers, bringing about the same problem discussed above.

As a result, clients are encouraged not to send the Proxy-Connection header field in any requests.

Clients are also encouraged to consider the use of Connection: keep-alive in requests carefully; while they can enable persistent connections with HTTP/1.0 servers, clients using them will need to monitor the connection for "hung" requests (which indicate that the client ought stop sending the header field), and this mechanism ought not be used by clients at all when a proxy is being used.

A.1.3. Introduction of Transfer-Encoding

HTTP/1.1 introduces the Transfer-Encoding header field (Section 3.3.1). Transfer codings need to be decoded prior to forwarding an HTTP message over a MIME-compliant protocol.

A.2. Changes from RFC 2616

HTTP's approach to error handling has been explained. (Section 2.5)

The HTTP-version ABNF production has been clarified to be case-sensitive. Additionally, version numbers have been restricted to single digits, due to the fact that implementations are known to handle multi-digit version numbers incorrectly. (Section 2.6)

Userinfo (i.e., username and password) are now disallowed in HTTP and HTTPS URIs, because of security issues related to their transmission on the wire. (Section 2.7.1)

The HTTPS URI scheme is now defined by this specification; previously, it was done in Section 2.4 of [RFC2818]. Furthermore, it implies end-to-end security. (Section 2.7.2)

HTTP messages can be (and often are) buffered by implementations; despite it sometimes being available as a stream, HTTP is fundamentally a message-oriented protocol. Minimum supported sizes for various protocol elements have been suggested, to improve interoperability. (Section 3)

Invalid whitespace around field-names is now required to be

され、応答の送信後にサーバーによって閉じられます。ただし、一部の実装では、[RFC2068]のセクション19.7.1で説明されている永続的な接続の明示的にネゴシエートされた（ "Keep-Alive"）バージョンを実装しています。

一部のクライアントとサーバーは、 "Connection：keep-alive"リクエストヘッダーフィールドを使用して明示的にネゴシエートすることにより、永続的な接続に対するこれらの以前のアプローチとの互換性を望む場合があります。ただし、HTTP / 1.0持続的接続の一部の実験的な実装には問題があります。たとえば、HTTP / 1.0プロキシサーバーが接続を認識しない場合、そのヘッダーフィールドは次の受信サーバーに誤って転送され、接続がハングします。

試みられた解決策の1つは、特にプロキシを対象としたProxy-Connectionヘッダーフィールドの導入でした。実際には、これも機能しませんでした。プロキシが複数のレイヤーに展開されることが多く、上記で説明した同じ問題が発生するためです。

その結果、クライアントは、どの要求でもProxy-Connectionヘッダーフィールドを送信しないことをお勧めします。

クライアントは、Connectionの使用を検討することもお勧めします。それらはHTTP / 1.0サーバーとの永続的な接続を有効にすることができますが、それらを使用するクライアントは「ハング」リクエスト（クライアントがヘッダーフィールドの送信を停止すべきであることを示す）について接続を監視する必要があり、このメカニズムはクライアントで使用されるべきではありません。プロキシが使用されている場合はすべて。

A.1.3. 転送エンコーディングの紹介

HTTP / 1.1では、Transfer-Encodingヘッダーフィールドが導入されています（セクション3.3.1）。 MIME準拠のプロトコルでHTTPメッセージを転送する前に、転送コーディングをデコードする必要があります。

A.2. RFC 2616からの変更

エラー処理に対するHTTPのアプローチについて説明しました。（セクション2.5）

HTTPバージョンのABNFプロダクションでは、大文字と小文字が区別されることが明確になっています。さらに、実装は複数桁のバージョン番号を誤って処理することが知られているため、バージョン番号は1桁に制限されています。 （セクション2.6）

Userinfo（つまり、ユーザー名とパスワード）は、HTTPおよびHTTPS URIでは許可されていません。これは、ネットワーク上での送信に関連するセキュリティ問題のためです。 （セクション2.7.1）

HTTPS URIスキームがこの仕様で定義されるようになりました。以前は、[RFC2818]のセクション2.4で行われていました。さらに、エンドツーエンドのセキュリティを意味します。 （セクション2.7.2）

HTTPメッセージは実装によってバッファリングできます（多くの場合はバッファリングされます）。ストリームとして利用できることもありますが、HTTPは基本的にメッセージ指向プロトコルです。相互運用性を向上させるために、さまざまなプロトコル要素でサポートされる最小サイズが提案されています。（セクション3）

フィールド名を囲む無効な空白は、受け入れるとセキュリティ

rejected, because accepting it represents a security vulnerability. The ABNF productions defining header fields now only list the field value. (Section 3.2)	の脆弱性を表すため、拒否する必要があります。ヘッダーフィールドを定義するABNFプロダクションは、フィールド値のみをリストするようになりました。（セクション3.2）
Rules about implicit linear whitespace between certain grammar productions have been removed; now whitespace is only allowed where specifically defined in the ABNF. (Section 3.2.3)	特定の文法プロダクション間の暗黙的な線形空白に関するルールが削除されました。現在、空白は、ABNFで明確に定義されている場合にのみ許可されます。（セクション3.2.3）
Header fields that span multiple lines ("line folding") are deprecated. (Section 3.2.4)	複数の行にまたがるヘッダーフィールド（「行の折り返し」）は非推奨になりました。（セクション3.2.4）
The NUL octet is no longer allowed in comment and quoted-string text, and handling of backslash-escaping in them has been clarified. The quoted-pair rule no longer allows escaping control characters other than HTAB. Non-US-ASCII content in header fields and the reason phrase has been obsoleted and made opaque (the TEXT rule was removed). (Section 3.2.6)	NULオクテットはコメントおよび引用文字列テキストで許可されなくなり、バックスラッシュエスケープの取り扱いが明確になりました。クォートペアルールでは、HTAB以外の制御文字をエスケープすることはできなくなりました。ヘッダーフィールドのUS-ASCII以外のコンテンツと理由フレーズが廃止され、不透明になりました（TEXTルールは削除されました）。（セクション3.2.6）
Bogus Content-Length header fields are now required to be handled as errors by recipients. (Section 3.3.2)	偽のContent-Lengthヘッダーフィールドは、受信者によってエラーとして処理される必要があります。（セクション3.3.2）
The algorithm for determining the message body length has been clarified to indicate all of the special cases (e.g., driven by methods or status codes) that affect it, and that new protocol elements cannot define such special cases. CONNECT is a new, special case in determining message body length. "multipart/byteranges" is no longer a way of determining message body length detection. (Section 3.3.3)	メッセージ本文の長さを決定するためのアルゴリズムは、それに影響するすべての特殊なケース（たとえば、メソッドまたはステータスコードによって駆動される）を示すように明確化されており、新しいプロトコル要素はそのような特殊なケースを定義できません。 CONNECTは、メッセージ本文の長さを決定する新しい特殊なケースです。 「multipart / byteranges」は、メッセージ本文の長さの検出を決定する方法ではなくなりました。（セクション3.3.3）
The "identity" transfer coding token has been removed. (Sections 3.3 and 4)	「ID」転送コーディングトークンが削除されました。（セクション3.3および4）
Chunk length does not include the count of the octets in the chunk header and trailer. Line folding in chunk extensions is disallowed. (Section 4.1)	チャンク長には、チャンクヘッダーとトレーラーのオクテットの数は含まれません。チャンク拡張での行折りたたみは許可されていません。（セクション4.1）
The meaning of the "deflate" content coding has been clarified. (Section 4.2.2)	「deflate」コンテンツコーディングの意味が明確になりました。（セクション4.2.2）
The segment + query components of RFC 3986 have been used to define the request-target, instead of abs_path from RFC 1808. The asterisk-form of the request-target is only allowed with the OPTIONS method. (Section 5.3)	RFC 1808のabs_pathの代わりに、RFC 3986のセグメント+クエリコンポーネントが要求ターゲットの定義に使用されています。要求ターゲットのアスタリスク形式は、OPTIONSメソッドでのみ使用できます。（セクション5.3）
The term "Effective Request URI" has been introduced. (Section 5.5)	「Effective Request URI」という用語が導入されました。（セクション5.5）
Gateways do not need to generate Via header fields anymore. (Section 5.7.1)	ゲートウェイは、Viaヘッダーフィールドを生成する必要がなくなりました。（セクション5.7.1）
Exactly when "close" connection options have to be sent has been clarified. Also, "hop-by-hop" header fields are required to appear in the Connection header field; just because they're defined as hop-by-hop in this specification doesn't exempt them. (Section 6.1)	「クローズ」接続オプションをいつ送信する必要があるかが明確になりました。また、「ホップバイホップ」ヘッダーフィールドは、接続ヘッダーフィールドに表示される必要があります。この仕様でホップバイホップとして定義されているからといって、それらは免除されません。（セクション6.1）
The limit of two connections per server has been removed. An idempotent sequence of requests is no longer required to be retried. The requirement to retry requests under certain circumstances when the server prematurely closes the connection has been removed. Also, some extraneous requirements about when servers are allowed to close connections prematurely have been removed. (Section 6.3)	サーバーごとの2つの接続の制限は削除されました。べき等の一連の要求を再試行する必要がなくなりました。サーバーが接続を時期尚早に閉じるときに、特定の状況下で要求を再試行する必要がなくなりました。また、サーバーが時期尚早に接続を閉じることが許可される時期に関するいくつかの無関係な要件が削除されました。（セクション6.3）

The semantics of the Upgrade header field is now defined in responses other than 101 (this was incorporated from [RFC2817]). Furthermore, the ordering in the field value is now significant. (Section 6.7)

Upgradeヘッダーフィールドのセマンティクスは、101以外の応答で定義されるようになりました（これは[RFC2817]から組み込まれました）。さらに、フィールド値の順序が重要になります。（6.7節）

Empty list elements in list productions (e.g., a list header field containing ", ,") have been deprecated. (Section 7)

リスト生成の空のリスト要素（たとえば、「、」を含むリストヘッダーフィールド）は非推奨になりました。（セクション7）

Registration of Transfer Codings now requires IETF Review (Section 8.4)

Transfer Codingsの登録にはIETFレビューが必要です（セクション8.4）

This specification now defines the Upgrade Token Registry, previously defined in Section 7.2 of [RFC2817]. (Section 8.6)

この仕様では、[RFC2817]のセクション7.2で以前に定義されたアップグレードトークンレジストリを定義するようになりました。（8.6節）

The expectation to support HTTP/0.9 requests has been removed. (Appendix A)

HTTP / 0.9リクエストをサポートするという期待は削除されました。（付録A）

Issues with the Keep-Alive and Proxy-Connection header fields in requests are pointed out, with use of the latter being discouraged altogether. (Appendix A.1.2)

リクエストのKeep-AliveおよびProxy-Connectionヘッダーフィールドの問題が指摘されていますが、後者の使用は完全に推奨されません。（付録A.1.2）

Appendix B. Collected ABNF

付録B.収集されたABNF

BWS = OWS

BWS = OWS

```
Connection = *( "," OWS ) connection-option *( OWS "," [ OWS
 connection-option ] )
```

Content-Length = 1*DIGIT

コンテンツの長さ= 1 * DIGIT

```
HTTP-message = start-line *( header-field CRLF ) CRLF [ message-body
 ]
HTTP-name = %x48.54.54.50 ; HTTP
HTTP-version = HTTP-name "/" DIGIT "." DIGIT
Host = uri-host [ ":" port ]
```

```
OWS = *( SP / HTAB )
```

```
RWS = 1*( SP / HTAB )
```

```
TE = [ ( "," / t-codings ) *( OWS "," [ OWS t-codings ] ) ]
Trailer = *( "," OWS ) field-name *( OWS "," [ OWS field-name ] )
Transfer-Encoding = *( "," OWS ) transfer-coding *( OWS "," [ OWS
 transfer-coding ] )
```

```
URI-reference = <URI-reference, see [RFC3986], Section 4.1>
Upgrade = *( "," OWS ) protocol *( OWS "," [ OWS protocol ] )
```

```
Via = *( "," OWS ) ( received-protocol RWS received-by [ RWS comment
 ] ) *( OWS "," [ OWS ( received-protocol RWS received-by [ RWS
 comment ] ) ] )
```

```
absolute-URI = <absolute-URI, see [RFC3986], Section 4.3>
absolute-form = absolute-URI
absolute-path = 1*( "/" segment )
asterisk-form = "*"
authority = <authority, see [RFC3986], Section 3.2>
authority-form = authority chunk = chunk-size [ chunk-ext ] CRLF chunk-data CRLF
chunk-data = 1*OCTET
chunk-ext = *( ";" chunk-ext-name [ "=" chunk-ext-val ] )
chunk-ext-name = token
chunk-ext-val = token / quoted-string
chunk-size = 1*HEXDIG
chunked-body = *chunk last-chunk trailer-part CRLF
comment = "(" *( ctext / quoted-pair / comment ) ")"
connection-option = token
ctext = HTAB / SP / %x21-27 ; '!'-'''
 / %x2A-5B ; '*'-'['
 / %x5D-7E ; ']'-'~'
```

```
 / obs-text
```

```
field-content = field-vchar [ 1*( SP / HTAB ) field-vchar ]
field-name = token
field-value = *( field-content / obs-fold )
field-vchar = VCHAR / obs-text
fragment = <fragment, see [RFC3986], Section 3.5>
```

```
header-field = field-name ":" OWS field-value OWS
http-URI = "http://" authority path-abempty [ "?" query ] [ "#"
 fragment ]
https-URI = "https://" authority path-abempty [ "?" query ] [ "#"
 fragment ]
```

last-chunk = 1*"0" [chunk-ext] CRLF

last-chunk = 1 * "0" [chunk-ext] CRLF

```
message-body = *OCTET
method = token
```

```
obs-fold = CRLF 1*( SP / HTAB )
obs-text = %x80-FF
origin-form = absolute-path [ "?" query ]
```

```
partial-URI = relative-part [ "?" query ]
path-abempty = <path-abempty, see [RFC3986], Section 3.3>
port = <port, see [RFC3986], Section 3.2.3>
protocol = protocol-name [ "/" protocol-version ]
protocol-name = token
protocol-version = token
pseudonym = token
```

```
qdtext = HTAB / SP / "!" / %x23-5B ; '#'-'['
 / %x5D-7E ; ']'-'~'
 / obs-text
query = <query, see [RFC3986], Section 3.4>
quoted-pair = "\" ( HTAB / SP / VCHAR / obs-text ) quoted-string = DQUOTE *( qdtext / quoted-pair ) DQUOTE
```

```
rank = ( "0" [ "." *3DIGIT ] ) / ( "1" [ "." *3"0" ] )
reason-phrase = *( HTAB / SP / VCHAR / obs-text )
received-by = ( uri-host [ ":" port ] ) / pseudonym
received-protocol = [ protocol-name "/" ] protocol-version
relative-part = <relative-part, see [RFC3986], Section 4.2>
request-line = method SP request-target SP HTTP-version CRLF
request-target = origin-form / absolute-form / authority-form /
 asterisk-form
```

```
scheme = <scheme, see [RFC3986], Section 3.1>
segment = <segment, see [RFC3986], Section 3.3>
start-line = request-line / status-line
status-code = 3DIGIT
status-line = HTTP-version SP status-code SP reason-phrase CRLF
```

```
t-codings = "trailers" / ( transfer-coding [ t-ranking ] )
t-ranking = OWS ";" OWS "q=" rank
tchar = "!" / "#" / "$" / "%" / "&" / "'" / "*" / "+" / "-" / "." /
 "^" / "_" / "`" / "|" / "~" / DIGIT / ALPHA
token = 1*tchar
trailer-part = *( header-field CRLF )
transfer-coding = "chunked" / "compress" / "deflate" / "gzip" /
 transfer-extension
transfer-extension = token *( OWS ";" OWS transfer-parameter )
transfer-parameter = token BWS "=" BWS ( token / quoted-string )
```

```
uri-host = <host, see [RFC3986], Section 3.2.2> Index
```

A absolute-form (of request-target) 42 accelerator 10 application/http Media Type 63 asterisk-form (of request-target) 43 authoritative response 67 authority-form (of request-target) 42-43

絶対形式（要求ターゲット）42アクセラレータ10アプリケーション/ httpメディアタイプ63アスタリスク形式（要求ターゲット）43信頼できる応答67権限形式（要求ターゲット）42-43

B browser 7

Bブラウザ7

Authors' Addresses

Roy T. Fielding (editor) Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA

```
  EMail: fielding@gbiv.com
  URI:   http://roy.gbiv.com/
```


著者のアドレス

ロイT.フィールディング（編集者）Adobe Systems Incorporated 345 Park Ave San Jose、CA 95110 USA

Julian F. Reschke (editor) greenbytes GmbH Hafenweg 16
Muenster, NW 48155 Germany

Julian F. Reschke（編集者）greenbytes GmbH Hafenweg 16
Muenster、NW 48155ドイツ

```
EMail: julian.reschke@greenbytes.de
URI:   http://greenbytes.de/tech/webdav/
```