Força-Tarefa de Engenharia da Internet (IETF)R. Fielding, Ed. Pedido de comentários: 7231Adobe
Obsoleto: 2616J. Reschke, Ed.
Atualizações: 2817 bytes verdes
Categoria: Padrões FaixaJunho 2014
ISSN: 2070-1721

Protocolo de Transferência de Hipertexto (HTTP/1.1):

Semântica e Resumo de Conteúdo

O protocolo HTTP (Hypertext Transfer Protocol) é um protocolo de nível de aplicativo sem estado para informações de hipertexto distribuídas e colaborativas Sistemas. Este documento define a semântica das mensagens HTTP/1.1, conforme expressa por métodos de solicitação, campos de cabeçalho de solicitação, códigos de status de resposta e campos de cabeçalho de resposta, juntamente com a carga útil de mensagens (metadados e conteúdo do corpo) e mecanismos para negociação de conteúdo.

Status deste memorando

Este é um documento do Internet Standards Track.

Este documento é um produto da Internet Engineering Task Force (IETF). Representa o consenso da comunidade IETF. Ele recebeu revisão pública e foi aprovado para publicação pelo Internet Engineering Steering Group (IESG). Mais informações sobre os Padrões da Internet estão disponíveis na Seção 2 da RFC 5741.

Informações sobre o status atual deste documento, qualquer errata e como fornecer feedback sobre ele podem ser obtidas em http://www.rfc-editor.org/info/rfc7231.

Aviso de direitos autorais

Índice

1. Introdução

Cada mensagem HTTP (Hypertext Transfer Protocol) é uma solicitação ou um resposta. Um servidor escuta uma solicitação em uma conexão, analisa cada mensagem recebida, interpreta a semântica da mensagem em relação ao destino da solicitação identificada e responde a essa solicitação com uma ou mais mensagens de resposta. Um cliente constrói mensagens de solicitação para comunicar intenções específicas, examina as respostas recebidas para ver se as intenções foram realizadas e determina como interpretar os resultados. Este documento define a semântica de solicitação e resposta HTTP/1.1 em termos da arquitetura definida em [RFC7230].

O HTTP fornece uma interface uniforme para interagir com um recurso (Seção 2), independentemente de seu tipo, natureza ou implementação, por meio da manipulação e transferência de representações (Seção 3).

A semântica HTTP inclui as intenções definidas por cada método de solicitação (Seção 4), extensões para a semântica que pode ser descrita nos campos de cabeçalho de solicitação (Seção 5), o significado de códigos de status para indicar uma resposta legível por máquina (Seção 6) e o significado de outros dados de controle e metadados de recursos que podem ser fornecidos nos campos de cabeçalho de resposta (Seção 7).

Este documento também define metadados de representação que descrevem como uma carga útil deve ser interpretada por um destinatário, os campos de cabeçalho da solicitação que podem influenciar a seleção de conteúdo e os vários algoritmos de seleção que são coletivamente chamados de "negociação de conteúdo" (Seção 3.4).

1.1. Conformidade e tratamento de erros

As palavras-chave "DEVE", "NÃO DEVE", "OBRIGATÓRIO", "DEVE", "NÃO DEVE", "DEVE", "NÃO DEVE", "RECOMENDADO", "PODE" e "OPCIONAL" neste documento devem ser interpretados conforme descrito em [RFC2119].

Os critérios de conformidade e as considerações relativas ao tratamento de erros são definidos na Secção 2.5 de [RFC7230].

1.2. Notação de sintaxe

Esta especificação usa a notação ABNF (Augmented Backus-Naur Form) de [RFC5234] com uma extensão de lista, definida na Seção 7 de [RFC7230], que permite a definição compacta de listas separadas por vírgulas usando um operador '#' (semelhante a como o operador '*' indica repetição). O Apêndice C descreve as regras importadas de outros documentos. O Apêndice D mostra a gramática coletada com todos os operadores de lista expandidos para a notação ABNF padrão.

Esta especificação usa os termos "caractere", "esquema de codificação de caracteres", "conjunto de caracteres" e "elemento de protocolo" conforme definidos em [RFC6365].

2. Recursos

O destino de uma solicitação HTTP é chamado de "recurso". O HTTP não limita a natureza de um recurso; ele apenas define uma interface que pode ser usada para interagir com recursos. Cada recurso é identificado por um Uniform Resource Identifier (URI), conforme descrito na Seção 2.7 de [RFC7230].

Quando um cliente constrói uma mensagem de solicitação HTTP/1.1, ele envia o URI de destino em uma das várias formas, conforme definido na (Seção 5.3 de [RFC7230]). Quando uma solicitação é recebida, o servidor reconstrói um URI de solicitação efetivo para o recurso de destino (Seção 5.5 de [RFC7230]).

Um objetivo de design do HTTP é separar a identificação de recursos da semântica de solicitação, o que é possível ao adquirir a semântica de solicitação no método de solicitação (Seção 4) e alguns campos de cabeçalho de modificação de solicitação (Seção 5). Se houver um conflito entre a semântica do método e qualquer semântica implícita no próprio URI, conforme descrito na Seção 4.2.1, a semântica do método terá precedência.

3. Representações

Considerando que um recurso pode ser qualquer coisa, e que a interface uniforme fornecida pelo HTTP é semelhante a uma janela através da qual se pode observar e agir sobre tal coisa apenas através da comunicação de mensagens a algum ator independente do outro lado, uma abstração é necessária para representar ("tomar o lugar") do estado atual ou desejado dessa coisa em nossas comunicações. Essa abstração é chamada de representação [REST].

Para fins de HTTP, uma "representação" é uma informação que se destina a refletir um estado passado, atual ou desejado de um determinado recurso, em um formato que pode ser prontamente comunicado por meio do protocolo e que consiste em um conjunto de metadados de representação e um fluxo potencialmente ilimitado de dados de representação.

Um servidor de origem pode ser fornecido ou ser capaz de gerar várias representações, cada uma destinada a refletir o estado atual de um recurso de destino. Nesses casos, algum algoritmo é usado pelo servidor de origem para selecionar uma dessas representações como mais aplicável a uma determinada solicitação, geralmente com base no conteúdo negociação. Esta "representação selecionada" é usada para fornecer o

dados e metadados para avaliar solicitações condicionais [RFC7232] e construir a carga útil para 200 (OK) e 304 (Não Modificado) respostas para GET (Seção 4.3.1).

3.1. Metadados de representação

Os campos de cabeçalho de representação fornecem metadados sobre a representação. Quando uma mensagem inclui um corpo de carga, os campos de cabeçalho de representação descrevem como interpretar os dados de representação incluídos no corpo da carga. Em resposta a uma solicitação HEAD, os campos de cabeçalho de representação descrevem os dados de representação que teriam sido incluídos no corpo da carga se a mesma solicitação fosse um GET.

Os seguintes campos de cabeçalho transmitem metadados de representação:

Nome do campo de cabeçalho	Definido em...
Tipo de conteúdo	Secção 3.1.1.5
Codificação de conteúdo	Secção 3.1.2.2
Conteúdo-Idioma	Secção 3.1.3.2
Localização do conteúdo	Secção 3.1.4.2

3.1.1. Processamento de dados de representação

3.1.1.1. Tipo de mídia

O HTTP usa os tipos de mídia da Internet [RFC2046] nos campos de cabeçalho Content-Type (Seção 3.1.1.5) e Accept (Seção 5.3.2) para fornecer digitação de dados aberta e extensível e negociação de tipo.
Os tipos de mídia definem um formato de dados e vários modelos de processamento: como processar esses dados de acordo com cada contexto em que são recebidos.

```
tipo de mídia = tipo "/" subtipo *( OWS ";" Parâmetro
OWS ) type= token
subtipo= token
```

O tipo/subtipo PODE ser seguido por parâmetros na forma de pares nome=valor.

```
parâmetro= token "=" ( token / string entre aspas )
```

Os tokens de nome, tipo, subtipo e parâmetro não diferenciam maiúsculas de minúsculas. Os valores de parâmetro podem ou não diferenciar maiúsculas de minúsculas, dependendo da semântica do parâmetro nome. A presença ou ausência de um parâmetro pode ser significativa para o processamento de um tipo de mídia, dependendo de sua definição no registro de tipo de mídia.

Um valor de parâmetro que corresponde à produção de token pode ser transmitido como um token ou dentro de um cadeia de caracteres entre aspas. Os valores entre aspas e sem aspas são equivalentes. Por exemplo, os exemplos a seguir são todos equivalentes, mas o primeiro é preferido para consistência:

```
texto/html; charset=utf-8
texto/html; charset=UTF-8
Texto/HTML; charset="utf-
8" texto/html;
charset="utf-8"
```

Os tipos de mídia da Internet devem ser registrados na IANA de acordo com os procedimentos definidos em [BCP13].

> Observação: ao contrário de algumas construções semelhantes em outros campos de cabeçalho, os parâmetros de tipo de mídia não permitem espaços em branco (mesmo espaços em branco "ruins") ao redor do caractere "=".

3.1.1.2. Conjunto de caracteres

O HTTP usa nomes de conjuntos de caracteres para indicar ou negociar o esquema de codificação de caracteres de uma representação textual [RFC6365]. Um conjunto de caracteres é identificado por um token que não diferencia maiúsculas de minúsculas.

```
conjunto de caracteres = token
```

Os nomes dos conjuntos de caracteres devem ser registrados no registro "Conjuntos de caracteres" da IANA (<http://www.iana.org/assignments/character-sets>) de acordo com os procedimentos definidos em [RFC2978].

3.1.1.3. Canonização e padrões de texto

Os tipos de mídia da Internet são registrados com uma forma canônica para serem interoperáveis entre sistemas com formatos de codificação nativos variados. As representações selecionadas ou transferidas via HTTP devem estar em forma canônica, por muitas das mesmas razões descritas pelo Multipurpose Internet Mail Extensions (MIME) [RFC2045]. No entanto, as características de desempenho das implantações de e-mail (ou seja, armazenar e encaminhar mensagens para pares) são significativamente diferentes daquelas comuns ao HTTP

e à Web (serviços de informações baseados em servidor).
Além disso, as restrições do MIME por uma questão de compatibilidade com protocolos de transferência de e-mail mais antigos não se aplicam ao HTTP (consulte o Apêndice A).

A forma canônica do MIME requer que os subtipos de mídia do tipo "texto" usem CRLF como a linha de texto quebrar. O HTTP permite a transferência de mídia de texto com CR ou LF simples representando apenas uma quebra de linha, quando essas quebras de linha são consistentes para um todo representação. Um remetente HTTP PODE gerar, e um destinatário DEVE ser capaz de analisar, quebras de linha em mídia de texto que consistem em CRLF, CR simples ou LF simples. Além disso, a mídia de texto em HTTP não se limita a conjuntos de caracteres que usam octetos 13 e 10 para CR e LF, respectivamente. Essa flexibilidade em relação a quebras de linha se aplica somente ao texto dentro de uma representação que recebeu um tipo de mídia "texto"; ele não se aplica a tipos "multipart" ou elementos HTTP fora do corpo da carga (por exemplo, campos de cabeçalho).

Se uma representação for codificada com uma codificação de conteúdo, os dados subjacentes devem estar em um formato definido acima antes de serem codificados.

3.1.1.4. Tipos de várias partes

O MIME fornece vários tipos de "várias partes" - encapsulamentos de uma ou mais representações em um único corpo de mensagem. Todos os tipos de várias partes compartilham uma sintaxe comum, conforme definido na Seção 5.1.1 de [RFC2046], e incluem um parâmetro de limite como parte do tipo de mídia valor. O corpo da mensagem é em si um elemento de protocolo; um remetente DEVE gerar apenas CRLF para representar quebras de linha entre partes do corpo.

O enquadramento de mensagem HTTP não usa o limite de várias partes como um indicador do comprimento do corpo da mensagem, embora possa ser usado por implementações que geram ou processam a carga. Por exemplo, o tipo "multipart/form-data" é frequentemente usado para transportar dados de formulário em uma solicitação, conforme descrito em [RFC2388], e o tipo "multipart/byteranges" é definido por esta especificação para uso em algumas respostas 206 (Conteúdo Parcial) [RFC7233].

3.1.1.5. Tipo de conteúdo

O campo de cabeçalho "Content-Type" indica o tipo de mídia da representação associada: a representação incluída na carga da mensagem ou a representação selecionada, conforme determinado pela semântica da mensagem. O tipo de mídia indicado define o formato de dados e como esses dados devem ser processados por um destinatário, dentro do escopo da semântica da mensagem recebida, depois que todas as codificações de conteúdo indicadas por Content-Encoding são decodificadas.

```
  Tipo de conteúdo = tipo de mídia
```

Os tipos de mídia são definidos na Seção 3.1.1.1. Um exemplo do campo é

Tipo de conteúdo: texto / html; conjunto de caracteres = ISO-8859-4

Um remetente que gera uma mensagem contendo um corpo de carga DEVE gerar um campo de cabeçalho Content-Type nessa mensagem, a menos que o tipo de mídia pretendido da representação incluída seja desconhecido para o remetente. Se um campo de cabeçalho Content-Type não estiver presente, o destinatário PODE assumir um tipo de mídia de "application/octet-stream" ([RFC2046], Seção 4.5.1) ou examinar os dados para determinar seu tipo.

Na prática, os proprietários de recursos nem sempre configuram corretamente seu servidor de origem para fornecer o Content-Type correto para uma determinada representação, com o resultado de que alguns clientes examinarão o conteúdo de uma carga e substituirão o especificado tipo. Os clientes que o fizerem correm o risco de tirar conclusões incorretas, o que pode expor riscos de segurança adicionais (por exemplo, "privilégio escalada"). Além disso, é impossível determinar a intenção do remetente examinando o formato dos dados: muitos formatos de dados correspondem a vários tipos de mídia que diferem apenas no processamento semântica. Os implementadores são incentivados a fornecer um meio de desabilitar essa "detecção de conteúdo" quando ela é usada.

3.1.2. Codificação para compactação ou integridade

3.1.2.1. Codificações de conteúdo

Os valores de codificação de conteúdo indicam uma transformação de codificação que foi ou pode ser aplicada a uma representação. As codificações de conteúdo são usadas principalmente para permitir que uma representação seja compactada ou transformada de forma útil sem perder a identidade de seu tipo de mídia subjacente e sem perda de informação. Freqüentemente, a representação é armazenada de forma codificada, transmitida diretamente e decodificada apenas pelo destinatário final.

codificação de conteúdo = token

Todos os valores de codificação de conteúdo não diferenciam maiúsculas de minúsculas e devem ser registrados no "Registro de codificação de conteúdo HTTP", conforme definido na Seção 8.4.Eles são usados nos campos de cabeçalho Accept-Encoding (Seção 5.3.4) e Content-Encoding (Seção 3.1.2.2).

Os seguintes valores de codificação de conteúdo são definidos por essa especificação:

compress (e x-compress): Consulte a Seção 4.2.1 de

[RFC7230]. deflate: Ver secção 4.2.2 de [RFC7230].

gzip (e x-gzip): Consulte a Seção 4.2.3 de [RFC7230].

3.1.2.2. Codificação de conteúdo

O campo de cabeçalho "Content-Encoding" indica quais codificações de conteúdo foram aplicadas à representação, além daquelas inerentes ao tipo de mídia e, portanto, quais mecanismos de decodificação devem ser aplicados para obter dados no tipo de mídia referenciado pelo campo de cabeçalho Content-Type. A codificação de conteúdo é usada principalmente para permitir que os dados de uma representação sejam compactados sem perder a identidade de seu tipo de mídia subjacente.

Codificação de conteúdo =

1#codificação de conteúdo Um exemplo

de seu uso é

Codificação de conteúdo: gzip

Se uma ou mais codificações tiverem sido aplicadas a uma representação, o remetente que aplicou as codificações DEVERÁ gerar um campo de cabeçalho Content-Encoding que liste as codificações de conteúdo na ordem em que foram aplicadas. Informações adicionais sobre os parâmetros de codificação podem ser fornecidas por outros campos de cabeçalho não definidos por esta especificação.

Ao contrário da Codificação de Transferência (Seção 3.3.1 de [RFC7230]), as codificações listadas em Codificação de Conteúdo são uma característica da representação; A representação é definida em termos da forma codificada e todos os outros metadados sobre a representação são sobre a forma codificada, salvo indicação em contrário na definição de metadados.
Normalmente, a representação só é decodificada pouco antes da renderização ou uso análogo.

Se o tipo de mídia incluir uma codificação inerente, como um formato de dados que está sempre compactado, essa codificação não será reformulada na codificação de conteúdo, mesmo que seja o mesmo algoritmo que um dos conteúdos Codings. Tal codificação de conteúdo só seria listada se, por algum motivo bizarro, fosse aplicada uma segunda vez para formar a representação. Da mesma forma, um servidor de origem pode optar por publicar os mesmos dados como várias representações que diferem apenas no fato de a codificação ser

definida como parte de Content-Type

ou Content-Encoding, uma vez que alguns agentes de usuário se comportarão de maneira diferente no tratamento de cada resposta (por exemplo, abrir um arquivo "Salvar como ..." em vez de descompactação automática e renderização de conteúdo).

Um servidor de origem PODE responder com um código de status de 415 (Tipo de Mídia Não Suportado) se uma representação na mensagem de solicitação tiver uma codificação de conteúdo que não seja aceitável.

3.1.3. Idioma do público

3.1.3.1. Tags de idioma

Uma etiqueta de idioma, conforme definido em [RFC5646], identifica uma linguagem natural falada, escrita ou transmitida por seres humanos para comunicação de informações a outros seres humanos. As linguagens de computador são explicitamente excluídas.

HTTP usa tags de idioma dentro do Accept-Language e Campos de cabeçalho Content-Language. Accept-Language usa a produção de intervalo de idioma mais ampla definida na Seção 5.3.5, enquanto Content-Language usa a produção de tags de idioma definida abaixo.

```
language-tag = <Language-Tag, consulte [RFC5646], Seção 2.1>
```

Uma marca de idioma é uma sequência de uma ou mais submarcas que não diferenciam maiúsculas de minúsculas, cada uma separada por um caractere de hífen ("-", %x2D). Na maioria dos casos, uma tag de idioma consiste em uma subtag de idioma principal que identifica uma ampla família de idiomas relacionados (por exemplo, "en" = inglês), que é opcionalmente seguida por uma série de subtags que refinam ou restringem o intervalo desse idioma (por exemplo, "en-CA" = a variedade do inglês conforme comunicado em Canadá). Espaços em branco não são permitidos em um idioma etiqueta. As tags de exemplo incluem:

```
fr, en-US, es-419, az-Arab, x-pig-latin, man-Nkoo-GN
```

Veja [RFC5646] para mais informações.

3.1.3.2. Linguagem do conteúdo

O campo de cabeçalho "Content-Language" descreve o(s) idioma(s) natural(is) do público-alvo para o representação. Observe que isso pode não ser equivalente a todos os idiomas usados na representação.

```
Idioma do conteúdo = 1#tag de idioma
```

As tags de idioma são definidas na Seção 3.1.3.1. O objetivo principal do Content-Language é permitir que um usuário identifique e diferencie representações de acordo com o idioma preferido do usuário.
Assim, se o conteúdo for destinado apenas a um público alfabetizado em dinamarquês, o campo apropriado é

```
Conteúdo-Idioma: da
```

Se nenhum Content-Language for especificado, o padrão será que o conteúdo seja destinado a todos os públicos de idioma. Isso pode significar que o remetente não o considera específico de nenhum idioma natural ou que o remetente não sabe para qual idioma se destina.

Vários idiomas PODEM ser listados para conteúdo destinado a vários públicos. Por exemplo, uma versão do "Tratado de Waitangi", apresentada simultaneamente nas versões originais em maori e inglês, exigiria

```
Conteúdo-Idioma: mi, en
```

No entanto, só porque vários idiomas estão presentes em uma representação não significa que ela se destina a vários públicos linguísticos. Um exemplo seria uma cartilha de linguagem para iniciantes, como "Uma Primeira Lição de Latim", que claramente se destina a ser usada por um público alfabetizado em inglês. Nesse caso, o Content-Language incluiria corretamente apenas "en".

Content-Language PODE ser aplicado a qualquer tipo de mídia - não se limita a documentos textuais.

3.1.4. Identificação

3.1.4.1. Identificando uma representação

Quando uma representação completa ou parcial é transferida em uma carga útil de mensagem, geralmente é desejável que o remetente forneça, ou o destinatário determine, um identificador para um recurso correspondente a essa representação.

Para uma mensagem de solicitação:

- Se a solicitação tiver um campo de cabeçalho Content-Location, o remetente declarará que a carga é uma representação do recurso identificado pelo valor do campo Content-Location. No entanto, tal afirmação não pode ser confiável, a menos que possa ser verificada por outros meios (não definidos por esta especificação). As informações ainda podem ser úteis para links de histórico de revisões.

- Caso contrário, a carga não será identificada.

Para uma mensagem de resposta, as seguintes regras são aplicadas em ordem até que uma correspondência seja encontrada:

1. Se o método de solicitação for GET ou HEAD e o código de status de resposta for 200 (OK), 204 (Sem Conteúdo), 206 (Conteúdo Parcial) ou 304 (Não Modificado), a carga será uma representação do recurso identificado pelo URI de solicitação efetivo (Seção 5.5 de [RFC7230]).

2. Se o método de solicitação for GET ou HEAD e o código de status de resposta for 203 (Informações Não Autoritativas), a carga será uma representação potencialmente modificada ou aprimorada do recurso de destino, conforme fornecido por um intermediário.

3. Se a resposta tiver um campo de cabeçalho Content-Location e seu valor de campo for uma referência ao mesmo URI que o URI de solicitação efetiva, a carga será uma representação do recurso identificado pelo URI de solicitação efetiva.

4. Se a resposta tiver um campo de cabeçalho Content-Location e seu valor de campo for uma referência a um URI diferente do URI de solicitação efetivo, o remetente declarará que a carga é uma representação do recurso identificado pelo Content-Location valor do campo. No entanto, tal afirmação não pode ser confiável, a menos que possa ser verificada por outros meios (não definidos por esta especificação).

5. Caso contrário, a carga não será identificada.

3.1.4.2. Localização do conteúdo

O campo de cabeçalho "Content-Location" faz referência a um URI que pode ser usado como um identificador para um recurso específico correspondente à representação na carga útil desta mensagem. Em outras palavras, se alguém executasse uma solicitação GET nesse URI no momento da geração dessa mensagem, uma resposta 200 (OK) conteria a mesma representação que está incluída como carga útil nessa mensagem.

```
Content-Location = absolute-URI / parcial-URI
```

O valor Content-Location não substitui o URI de solicitação efetivo (Seção 5.5 de [RFC7230]). São metadados de representação. Ele tem a mesma sintaxe e semântica que o campo de cabeçalho de mesmo nome definido para partes do corpo MIME na Seção 4 de [RFC2557]. No entanto, sua aparência em uma mensagem HTTP tem algumas implicações especiais para os destinatários HTTP.

Se Content-Location estiver incluído em uma mensagem de resposta 2xx (bem-sucedida) e seu valor se referir (após a conversão para a forma absoluta) a um URI que seja o mesmo que o URI de solicitação efetiva, o destinatário PODERÁ considerar a carga útil como uma representação atual desse recurso no momento indicado pela data de origem da mensagem. Para uma solicitação GET (Seção 4.3.1) ou HEAD (Seção 4.3.2), isso é o mesmo que a semântica padrão quando nenhum Content-Location é fornecido pelo servidor. Para uma solicitação de mudança de estado como PUT (Seção 4.3.4) ou POST (Seção 4.3.3), isso implica que a resposta do servidor contém a nova representação desse recurso, distinguindo-o assim das representações que podem relatar apenas sobre a ação (por exemplo, "Ele funcionou!"). Isso permite que os aplicativos de criação atualizem suas cópias locais sem a necessidade de uma solicitação GET subsequente.

Se Content-Location estiver incluído em uma mensagem de resposta 2xx (bem-sucedida) e seu valor de campo se referir a um URI diferente do URI de solicitação efetivo, o servidor de origem alegará que o URI é um identificador para um recurso diferente correspondente à representação incluída. Essa declaração só poderá ser confiável se ambos os identificadores compartilharem o mesmo proprietário do recurso, que não pode ser determinado programaticamente via HTTP.

- Para uma resposta a uma solicitação GET ou HEAD, essa é uma indicação de que o URI de solicitação efetiva se refere a um recurso sujeito à negociação de conteúdo e ao Content-Location field-value é um identificador mais específico para a representação selecionada.

- Para uma resposta 201 (Created) a um método de alteração de estado, um valor de campo Content-Location idêntico ao valor de campo Location indica que essa carga é uma representação atual do recurso recém-criado.

- Caso contrário, esse Content-Location indica que essa carga é uma representação que relata o status da ação solicitada e que o mesmo relatório está disponível (para acesso futuro com GET) no URI fornecido. Por exemplo, uma transação de compra feita por meio de uma solicitação POST pode incluir um documento de recebimento como o conteúdo da resposta 200 (OK); o valor do campo Content-Location fornece um identificador para recuperar uma cópia desse mesmo recibo no futuro.

Um agente de usuário que envia Content-Location em uma mensagem de solicitação está declarando que seu valor se refere a onde o agente de usuário obteve originalmente o conteúdo da representação anexa (antes de qualquer modificação feita por esse agente de usuário). Em outras palavras, o agente do usuário está fornecendo um link de retorno para a origem da representação original.

Um servidor de origem que recebe um campo Content-Location em uma mensagem de solicitação DEVE tratar as informações como contexto de solicitação transitória em vez de metadados a serem salvos literalmente como parte da representação. Um servidor de origem PODE usar esse contexto para orientar o processamento da solicitação ou salvá-la para outros usos, como links de origem ou controle de versão metadados. No entanto, um servidor de origem NÃO DEVE usar essas informações de contexto para alterar a semântica da solicitação.

Por exemplo, se um cliente fizer uma solicitação PUT em um recurso negociado e o servidor de origem aceitar esse PUT (sem redirecionamento), espera-se que o novo estado desse recurso seja consistente com a representação fornecida nesse PUT; o Content-Location não pode ser usado como uma forma de identificador de seleção de conteúdo reverso para atualizar apenas uma das representações negociadas. Se o agente do usuário quisesse a última semântica, ele teria aplicado o PUT diretamente ao URI do local do conteúdo.

3.2. Dados de representação

Os dados de representação associados a uma mensagem HTTP são fornecidos como o corpo da carga útil da mensagem ou referenciados pela semântica da mensagem e pelo URI de solicitação efetivo. Os dados de representação estão em um formato e codificação definidos pelos campos de cabeçalho de metadados de representação.

O tipo de dados dos dados de representação é determinado por meio dos campos de cabeçalho Content-Type e Content-Encoding. Eles definem um modelo de codificação ordenado de duas camadas:

```
dados de representação := Content-Encoding( Content-Type( bits ) )
```

3.3. Semântica de carga útil

Algumas mensagens HTTP transferem uma representação completa ou parcial como a mensagem "carga útil". Em alguns casos, uma carga pode conter apenas os campos de cabeçalho da representação associada (por exemplo, respostas a HEAD) ou apenas algumas partes dos dados de representação (por exemplo, o código de status 206 (Conteúdo parcial)).

A finalidade de uma carga em uma solicitação é definida pela semântica do método. Por exemplo, uma representação na carga útil de uma solicitação PUT (Seção 4.3.4) representa o estado desejado do recurso de destino se a solicitação for aplicada com êxito, enquanto uma representação na carga útil de uma solicitação POST (Seção 4.3.3) representa informações a serem processadas pelo recurso de destino.

Em uma resposta, a finalidade da carga é definida pelo método de solicitação e pelo status da resposta código. Por exemplo, a carga útil de um
200 (OK) resposta a GET (Seção 4.3.1) representa o estado atual do recurso de destino, conforme observado no momento da data de origem da mensagem (Seção 7.1.1.2), enquanto a carga útil do mesmo código de status em uma resposta ao POST pode representar o resultado do processamento ou o novo estado do recurso de destino após a aplicação do processamento. As mensagens de resposta com um código de status de erro geralmente contêm uma carga que representa a condição de erro, de modo que descreva o estado do erro e quais próximas etapas são sugeridas para resolvê-lo.

Os campos de cabeçalho que descrevem especificamente a carga, em vez da representação associada, são chamados de "cabeçalho de carga campos". Os campos de cabeçalho de carga útil são definidos em outras partes desta especificação, devido ao seu impacto na análise de mensagens.

Nome do campo de cabeçalho	Definido em...
Comprimento do conteúdo	Secção 3.3.2 de [RFC7230]
Faixa de conteúdo	Seção 4.2 de [RFC7233]
Reboque	A secção 4.4 do [RFC7230]
Codificação de transferência	Secção 3.3.1 de [RFC7230]

3.4. Negociação de conteúdo

Quando as respostas transmitem informações de carga, seja indicando um sucesso ou um erro, o servidor de origem geralmente tem maneiras diferentes de representar essas informações; por exemplo, em diferentes formatos, idiomas ou Codificações. Da mesma forma, diferentes usuários ou agentes de usuário podem ter diferentes capacidades, características ou preferências que podem influenciar qual representação, entre as disponíveis, seria melhor oferecer. Por esse motivo, o HTTP fornece mecanismos para negociação de conteúdo.

Essa especificação define dois padrões de negociação de conteúdo que podem ser tornados visíveis dentro do protocolo: "proativo", em que o servidor seleciona a representação com base nas preferências declaradas do agente do usuário, e negociação "reativa", em que o servidor fornece uma lista de representações para o agente do usuário escolher De. Outros padrões de negociação de conteúdo incluem "conteúdo condicional", em que a representação consiste em várias partes que são renderizadas seletivamente com base nos parâmetros do agente do usuário, "conteúdo ativo", em que a representação contém um script que faz solicitações adicionais (mais específicas) com base nas características do agente do usuário, e "Negociação de Conteúdo Transparente" ([RFC2295]), em que o conteúdo

seleção é realizada por um intermediário. Esses padrões não são mutuamente exclusivos e cada um tem compensações em aplicabilidade e praticidade.

Observe que, em todos os casos, o HTTP não está ciente da semântica do recurso. A consistência com que um servidor de origem responde às solicitações, ao longo do tempo e nas várias dimensões da negociação de conteúdo e, portanto, a "semelhança" das representações observadas de um recurso ao longo do tempo, é determinada inteiramente por qualquer entidade ou algoritmo que selecione ou gere essas respostas. HTTP não presta atenção ao homem por trás da cortina.

3.4.1. Negociação proativa

Quando as preferências de negociação de conteúdo são enviadas pelo agente do usuário em uma solicitação para incentivar um algoritmo localizado no servidor a selecionar a representação preferencial, isso é chamado de negociação proativa (também conhecida como negociação orientada pelo servidor). A seleção é baseada nas representações disponíveis para uma resposta (as dimensões sobre as quais ela pode variar, como idioma, codificação de conteúdo, etc.) em comparação com várias informações fornecidas na solicitação, incluindo os campos de negociação explícitos da Seção 5.3 e características implícitas, como o endereço de rede do cliente ou partes do campo User-Agent.

A negociação proativa é vantajosa quando o algoritmo para selecionar entre as representações disponíveis é difícil de descrever para um agente do usuário, ou quando o servidor deseja enviar seu "melhor palpite" para o agente do usuário junto com a primeira resposta (na esperança de evitar o atraso de ida e volta de uma solicitação subsequente se o "melhor palpite" for bom o suficiente para o usuário). Para melhorar o palpite do servidor, um agente de usuário PODE enviar campos de cabeçalho de solicitação que descrevem suas preferências.

A negociação proativa tem sérias desvantagens:

- É impossível para o servidor determinar com precisão o que pode ser "melhor" para qualquer usuário, uma vez que isso exigiria conhecimento completo dos recursos do agente do usuário e do uso pretendido para a resposta (por exemplo, o usuário deseja visualizá-lo na tela ou imprimi-lo em papel?);

- Fazer com que o agente do usuário descreva seus recursos em cada solicitação pode ser muito ineficiente (dado que apenas uma pequena porcentagem das respostas tem várias representações) e um risco potencial para a privacidade do usuário;

- Complica a implementação de um servidor de origem e os algoritmos para gerar respostas a uma solicitação; e

- Ele limita a reutilização de respostas para cache compartilhado.

Um agente de usuário não pode contar com que as preferências de negociação proativa sejam consistentemente respeitadas, pois o servidor de origem pode não implementar a negociação proativa para o recurso solicitado ou pode decidir que enviar uma resposta que não esteja em conformidade com as preferências do agente de usuário é melhor do que enviar uma resposta 406 (Não Aceitável).

Um campo de cabeçalho Vary (Seção 7.1.4) geralmente é enviado em uma resposta sujeita a negociação proativa para indicar quais partes das informações da solicitação foram usadas no algoritmo de seleção.

3.4.2. Negociação reativa

Com a negociação reativa (também conhecida como negociação orientada por agente), a seleção da melhor representação de resposta (independentemente do código de status) é realizada pelo agente do usuário após receber uma resposta inicial do servidor de origem que contém uma lista de recursos para Representações. Se o agente do usuário não estiver satisfeito com a representação de resposta inicial, ele poderá executar uma solicitação GET em um ou mais dos recursos alternativos, selecionados com base nos metadados incluídos na lista, para obter uma forma diferente de representação para essa resposta. A seleção de alternativas pode ser realizada automaticamente pelo agente do usuário ou manualmente pelo usuário selecionando em um menu gerado (possivelmente hipertexto).

Observe que o acima se refere a representações da resposta, em geral, não a representações do recurso. As representações alternativas só são consideradas representações do recurso de destino se a resposta na qual essas alternativas são fornecidas tiver a semântica de ser uma representação do recurso de destino (por exemplo, uma resposta 200 (OK) a uma solicitação GET) ou tiver a semântica de fornecer links para representações alternativas para o recurso de destino (por exemplo, uma resposta 300 (Múltipla Escolha) a uma solicitação GET).

Um servidor pode optar por não enviar uma representação inicial, diferente da lista de alternativas, e, assim, indicar que a negociação reativa pelo agente do usuário é preferida. Por exemplo, as alternativas listadas nas respostas com o 300 (Múltipla escolha) e Os códigos de status 406 (Não Aceitável) incluem informações sobre as representações disponíveis para que o usuário ou agente do usuário possa reagir fazendo uma seleção.

A negociação reativa é vantajosa quando a resposta varia em relação às dimensões comumente usadas (como tipo, idioma ou codificação), quando o servidor de origem não consegue determinar os recursos de um agente de usuário examinando a solicitação e, geralmente, quando caches públicos são usados para distribuir a carga do servidor e

reduzir o uso da rede.

A negociação reativa sofre com as desvantagens de transmitir uma lista de alternativas ao agente do usuário, o que degrada a latência percebida pelo usuário se transmitida na seção de cabeçalho e precisar de uma segunda solicitação para obter uma alternativa representação. Além disso, esta especificação não define um mecanismo para apoiar a seleção automática, embora não impeça que tal mecanismo seja desenvolvido como uma extensão.

4. Métodos de solicitação

4.1. Visão geral

O token do método de solicitação é a principal fonte de semântica de solicitação; Indica a finalidade para a qual o cliente fez essa solicitação e o que é esperado pelo cliente como um resultado bem-sucedido.

A semântica do método de solicitação pode ser ainda mais especializada pela semântica de alguns campos de cabeçalho quando presentes em uma solicitação (Seção 5) se essa semântica adicional não entrar em conflito com o método. Por exemplo, um cliente pode enviar campos de cabeçalho de solicitação condicional (Seção 5.2) para condicionar a ação solicitada ao estado atual do recurso de destino ([RFC7232]).

```
método = token
```

O HTTP foi originalmente projetado para ser utilizável como uma interface para sistemas de objetos distribuídos. O método de solicitação foi concebido como aplicando semântica a um recurso de destino da mesma forma que invocar um método definido em um objeto identificado aplicaria semântica. O token de método diferencia maiúsculas de minúsculas porque pode ser usado como um gateway para sistemas baseados em objeto com nomes de método que diferenciam maiúsculas de minúsculas.

Ao contrário dos objetos distribuídos, os métodos de solicitação padronizados em HTTP não são específicos do recurso, pois as interfaces uniformes fornecem melhor visibilidade e reutilização em sistemas baseados em rede [REST]. Uma vez definido, um método padronizado deve ter a mesma semântica quando aplicado a qualquer recurso, embora cada recurso determine por si mesmo se essa semântica é implementada ou permitida.

Essa especificação define vários métodos padronizados que são comumente usados em HTTP, conforme descrito a seguir mesa. Por convenção, os métodos padronizados são definidos em letras maiúsculas
Letras US-ASCII.

Método	Descrição	Sec.
GET	Transferir uma representação atual do destino	4.3.1
CABEÇA	O mesmo que GET, mas apenas transferir o status linha e cabeçalho	4.3.2
POSTAGEM	Executar o processamento específico do recurso em pedir carga útil.	4.3.3
PUT	Substitua todas as representações atuais de target com a solicitação carga útil.	4.3.4
EXCLUIR	Remova todas as representações atuais de alvo recurso.	4.3.5
CONECTAR	Estabelecer um túnel para o servidor identificado por O alvo recurso.	4.3.6
OPÇÕES	Descreva as opções de comunicação para o alvo recurso.	4.3.7
RASTREAMENTO	Executar um teste de loopback de mensagem ao longo do caminho para o destino recurso.	4.3.8

Todos os servidores de uso geral DEVEM suportar os métodos GET e HEAD. Todos os outros métodos são OPCIONAIS.

Métodos adicionais, fora do escopo desta especificação, foram padronizados para uso em HTTP. Todos esses métodos devem ser registrados no "Registro de Método do Protocolo de Transferência de Hipertexto (HTTP)" mantido pela IANA, conforme definido na Seção 8.1.

O conjunto de métodos permitidos por um recurso de destino pode ser listado em um campo de cabeçalho Permitir (Seção 7.4.1). No entanto, o conjunto de métodos permitidos pode mudar dinamicamente. Quando um método de solicitação é recebido que não é reconhecido ou não implementado por um servidor de origem, o servidor de origem DEVE responder com o código de status 501 (Não Implementado). Quando é recebido um método de solicitação conhecido por um servidor de origem, mas não permitido para o recurso de destino, o servidor de origem DEVE responder com o código de status 405 (Método Não Permitido).

4.2. Propriedades comuns do método

4.2.1. Métodos seguros

Os métodos de solicitação são considerados "seguros" se sua semântica definida for essencialmente somente leitura; ou seja, o cliente não solicita e não espera nenhuma alteração de estado no servidor de origem como resultado da aplicação de um método seguro a um recurso de destino. Da mesma forma, não se espera que o uso razoável de um método seguro cause nenhum dano, perda de propriedade ou ônus incomum ao servidor de origem.

Essa definição de métodos seguros não impede que uma implementação inclua um comportamento potencialmente prejudicial, que não seja totalmente somente leitura ou que cause efeitos colaterais ao invocar um método seguro. O que é importante, no entanto, é que o cliente não solicitou esse comportamento adicional e não pode ser responsabilizado por isso. Por exemplo, a maioria dos servidores acrescenta informações de solicitação para acessar arquivos de log na conclusão de cada resposta, independentemente do método, e isso é considerado seguro, mesmo que o armazenamento de log possa ficar cheio e travar o servidor. Da mesma forma, uma solicitação segura iniciada pela seleção de um anúncio na Web geralmente terá o efeito colateral de cobrar uma conta de publicidade.

Dos métodos de solicitação definidos por essa especificação, os métodos GET, HEAD, OPTIONS e TRACE são definidos como seguros.

O objetivo de distinguir entre métodos seguros e inseguros é permitir que processos automatizados de recuperação (spiders) e otimização de desempenho de cache (pré-busca) funcionem sem medo de causar danos. Além disso, permite que um agente de usuário aplique restrições apropriadas ao uso automatizado de métodos não seguros ao processar conteúdo potencialmente não confiável.

Um agente de usuário DEVE distinguir entre métodos seguros e não seguros ao apresentar ações potenciais a um usuário, de modo que o usuário possa ser informado de uma ação não segura antes que ela seja solicitada.

Quando um recurso é construído de forma que os parâmetros dentro do URI de solicitação efetivo tenham o efeito de selecionar uma ação, é responsabilidade do proprietário do recurso garantir que a ação seja consistente com a semântica do método de solicitação. Por exemplo, é comum que softwares de edição de conteúdo baseados na Web usem ações dentro de parâmetros de consulta, como "page?do=delete". Se a finalidade desse recurso for executar uma ação não segura, o proprietário do recurso DEVERÁ desabilitar ou proibir essa ação quando ela for acessada usando um método de solicitação segura. Não fazer isso resultará em efeitos colaterais infelizes quando os processos automatizados executarem um GET em cada referência de URI para fins de manutenção de link, pré-busca, criação de um índice de pesquisa, etc.

4.2.2. Métodos idempotentes

Um método de solicitação é considerado "idempotente" se o efeito pretendido no servidor de várias solicitações idênticas com esse método for o mesmo que o efeito de uma única solicitação. Dos métodos de solicitação definidos por essa especificação, os métodos de solicitação PUT, DELETE e safe são idempotentes.

Assim como a definição de seguro, a propriedade idempotente se aplica apenas ao que foi solicitado pelo usuário; Um servidor é livre para registrar cada solicitação separadamente, manter um histórico de controle de revisão ou implementar outros efeitos colaterais não idempotentes para cada solicitação idempotente.

Os métodos idempotentes são diferenciados porque a solicitação pode ser repetida automaticamente se ocorrer uma falha de comunicação antes que o cliente possa ler a resposta do servidor. Por exemplo, se um cliente enviar uma solicitação PUT e a conexão subjacente for fechada antes que qualquer resposta seja recebida, o cliente poderá estabelecer uma nova conexão e repetir a solicitação idempotente. Ele sabe que repetir a solicitação terá o mesmo efeito pretendido, mesmo que a solicitação original tenha sido bem-sucedida, embora a resposta possa ser diferente.

4.2.3. Métodos armazenáveis em cache

Os métodos de solicitação podem ser definidos como "armazenáveis em cache" para indicar que as respostas a eles podem ser armazenadas para reutilização futura; Para requisitos específicos, consulte [RFC7234]. Em geral, os métodos seguros que não dependem de uma resposta atual ou autoritativa são definidos como armazenáveis em cache; essa especificação define GET, HEAD e POST como armazenáveis em cache, embora a esmagadora maioria das implementações de cache dê suporte apenas a GET e HEAD.

4.3. Definições de método

4.3.1. OBTER

O método GET solicita a transferência de uma representação selecionada atual para o recurso de destino. O GET é o principal mecanismo de recuperação de informações e o foco de quase todas as otimizações de desempenho.
Portanto, quando as pessoas falam em recuperar algumas informações identificáveis via HTTP, geralmente estão se referindo a fazer uma solicitação GET.

É tentador pensar em identificadores de recursos como nomes de caminho de sistemas de arquivos remotos e em representações como sendo uma cópia do conteúdo de tais arquivos. Na verdade, é assim que muitos recursos são implementados (consulte a Seção 9.1 para considerações de segurança relacionadas). No entanto, não existem tais limitações em prática. A interface HTTP de um recurso tem a mesma probabilidade de ser implementada como uma árvore de objetos de conteúdo, uma exibição programática em vários registros de banco de dados ou um gateway para outras informações Sistemas. Mesmo quando o mecanismo de mapeamento de URI está vinculado a um sistema de arquivos, um servidor de origem pode ser configurado para executar os arquivos com a solicitação como entrada e enviar a saída como a

representação, em vez de transferir os arquivos diretamente. Independentemente disso, apenas o servidor de origem precisa saber como cada um de seus recursos

identifiers corresponde a uma implementação e como cada implementação gerencia selecionar e enviar uma representação atual do recurso de destino em uma resposta a GET.

Um cliente pode alterar a semântica de GET para ser uma "solicitação de intervalo", solicitando a transferência de apenas algumas partes da representação selecionada, enviando um campo de cabeçalho Range na solicitação ([RFC7233]).

Uma carga útil dentro de uma mensagem de solicitação GET não tem semântica definida; enviar um corpo de carga útil em uma solicitação GET pode fazer com que algumas implementações existentes rejeitem a solicitação.

A resposta a uma solicitação GET pode ser armazenada em cache; um cache PODE usá-lo para satisfazer solicitações GET e HEAD subsequentes, a menos que indicado de outra forma pelo campo de cabeçalho Cache-Control (Seção 5.2 de [RFC7234]).

4.3.2. CABEÇA

O método HEAD é idêntico ao GET, exceto que o servidor NÃO DEVE enviar um corpo de mensagem na resposta (ou seja, a resposta termina no final da seção de cabeçalho). O servidor DEVE enviar os mesmos campos de cabeçalho em resposta a uma solicitação HEAD que teria enviado se a solicitação fosse um GET, exceto que os campos de cabeçalho de carga útil (Seção 3.3) PODEM ser omitidos. Esse método pode ser usado para obter metadados sobre a representação selecionada sem transferir os dados de representação e é frequentemente usado para testar links de hipertexto para validade, acessibilidade e modificação recente.

Uma carga útil dentro de uma mensagem de solicitação HEAD não tem semântica definida; enviar um corpo de carga útil em uma solicitação HEAD pode fazer com que algumas implementações existentes rejeitem a solicitação.

A resposta a uma solicitação HEAD pode ser armazenada em cache; um cache PODE usá-lo para satisfazer solicitações HEAD subsequentes, a menos que indicado de outra forma pelo campo de cabeçalho Cache-Control (Seção 5.2 de [RFC7234]). Uma resposta HEAD também pode ter um efeito sobre as respostas armazenadas em cache anteriormente para GET; ver secção 4.3.5 da [RFC7234].

4.3.3. POSTAR

O método POST solicita que o recurso de destino processe a representação incluída na solicitação de acordo com a semântica específica do próprio recurso. Por exemplo, POST é usado para as seguintes funções (entre outras):

- Fornecer um bloco de dados, como os campos inseridos em um formulário HTML, para um processo de tratamento de dados;

- Postar uma mensagem em um quadro de avisos, grupo de notícias, lista de discussão, blog ou grupo de artigos semelhantes;

- Criar um novo recurso que ainda não foi identificado pelo servidor de origem; e

- Anexar dados às representações existentes de um recurso.

Um servidor de origem indica a semântica de resposta escolhendo um código de status apropriado, dependendo do resultado do processamento da solicitação POST; quase todos os códigos de status definidos por esta especificação podem ser recebidos em uma resposta ao POST (as exceções são 206 (Conteúdo parcial), 304 (Não modificado) e 416 (Intervalo não satisfatível)).

Se um ou mais recursos tiverem sido criados no servidor de origem como resultado do processamento bem-sucedido de uma solicitação POST, o servidor de origem DEVERÁ enviar uma resposta 201 (Criada) contendo um campo de cabeçalho Local que forneça um identificador para o recurso primário criado (Seção 7.1.2) e uma representação que descreva o status da solicitação ao se referir ao(s) novo(s) recurso(s).

As respostas às solicitações POST só podem ser armazenadas em cache quando incluem informações explícitas de atualização (consulte a Seção 4.2.1 de [RFC7234]). No entanto, o cache POST não é amplamente Implementado. Para os casos em que um servidor de origem deseja que o cliente seja capaz de armazenar em cache o resultado de um POST de uma forma que possa ser reutilizada por um GET posterior, o servidor de origem PODE enviar uma resposta 200 (OK) contendo o resultado e um
Content-Location que tem o mesmo valor que o URI de solicitação efetiva do POST (Seção 3.1.4.2).

Se o resultado do processamento de um POST for equivalente a uma representação de um recurso existente, um servidor de origem PODERÁ redirecionar o agente do usuário para esse recurso enviando uma resposta 303 (Ver Outro) com o identificador do recurso existente no Local campo. Isso tem os benefícios de fornecer ao agente do usuário um identificador de recurso e transferir a representação por meio de um método mais passível de cache compartilhado, embora ao custo de uma solicitação extra se o agente do usuário ainda não tiver a representação armazenada em cache.

4.3.4. PÔR

O método PUT solicita que o estado do recurso de destino seja criado ou substituído pelo estado definido pela representação incluída na mensagem de solicitação carga útil. Um PUT bem-sucedido de uma determinada representação sugeriria que um GET subsequente nesse mesmo recurso de destino resultará no envio de uma representação

equivalente em um 200 (OK) resposta. No entanto, não há garantia de que

essa alteração de estado será observável, pois o recurso de destino pode ser acionado por outros agentes de usuário em paralelo ou pode estar sujeito a processamento dinâmico pelo servidor de origem, antes que qualquer GET subsequente seja recebido. Uma resposta bem-sucedida implica apenas que a intenção do agente do usuário foi alcançada no momento de seu processamento pelo servidor de origem.

Se o recurso de destino não tiver uma representação atual e o PUT criar uma com êxito, o servidor de origem DEVERÁ informar o agente do usuário enviando uma resposta 201 (Criada). Se o recurso de destino tiver uma representação atual e essa representação for modificada com êxito de acordo com o estado da representação incluída, o servidor de origem DEVERÁ enviar uma resposta 200 (OK) ou 204 (Sem Conteúdo) para indicar a conclusão bem-sucedida da solicitação.

Um servidor de origem DEVE ignorar campos de cabeçalho não reconhecidos recebidos em uma solicitação PUT (ou seja, não os salve como parte do estado do recurso).

Um servidor de origem DEVE verificar se a representação PUT é consistente com quaisquer restrições que o servidor tenha para o recurso de destino que não podem ou não serão alteradas pelo PUT. Isso é particularmente importante quando o servidor de origem usa informações de configuração interna relacionadas ao URI para definir os valores para metadados de representação em respostas GET. Quando uma representação PUT é inconsistente com o recurso de destino, o servidor de origem DEVE torná-la consistente, transformando a representação ou alterando a configuração do recurso, ou responder com uma mensagem de erro apropriada contendo informações suficientes para explicar por que a representação é inadequado. Os códigos de status 409 (Conflito) ou 415 (Tipo de Mídia Sem Suporte) são sugeridos, sendo o último específico para restrições em valores de Tipo de Conteúdo.

Por exemplo, se o recurso de destino estiver configurado para sempre ter um Content-Type de "text/html" e a representação que está sendo PUT tiver um Content-Type de "image/jpeg", o servidor de origem deverá fazer um dos seguintes:

a. reconfigure o recurso de destino para refletir o novo tipo de mídia;

b. transformar a representação PUT em um formato consistente com o do recurso antes de salvá-la como o novo estado do recurso; ou

c. rejeite a solicitação com uma resposta 415 (Tipo de mídia não suportado) indicando que o recurso de destino está limitado a "text/html", talvez incluindo um link para um recurso diferente que seria um destino adequado para a nova representação.

O HTTP não define exatamente como um método PUT afeta o estado de um servidor de origem além do que pode ser expresso pela intenção da solicitação do agente do usuário e pela semântica do servidor de origem resposta. Ele não define o que um recurso pode ser, em nenhum sentido da palavra, além da interface fornecida via HTTP. Ele não define como o estado do recurso é "armazenado", nem como esse armazenamento pode ser alterado como resultado de uma alteração no estado do recurso, nem como o servidor de origem converte o estado do recurso em representações. De modo geral, todos os detalhes de implementação por trás da interface de recursos são intencionalmente ocultos pelo servidor.

Um servidor de origem NÃO DEVE enviar um campo de cabeçalho do validador (Seção 7.2), como um campo ETag ou Last-Modified, em um
resposta bem-sucedida ao PUT, a menos que os dados de representação da solicitação tenham sido salvos sem qualquer transformação aplicada ao corpo (ou seja, os novos dados de representação do recurso são idênticos aos dados de representação recebidos na solicitação PUT) e o valor do campo validador reflete o novo representação. Esse requisito permite que um agente de usuário saiba quando o corpo de representação que ele possui na memória permanece atual como resultado do PUT, portanto, não precisa ser recuperado novamente do servidor de origem, e que o(s) novo(s) validador(es) recebido(s) na resposta pode(m) ser usado(s) para futuras solicitações condicionais a fim de evitar substituições acidentais (Seção 5.2).

A diferença fundamental entre os métodos POST e PUT é destacada pela intenção diferente para a representação incluída. O recurso de destino em uma solicitação POST destina-se a lidar com a representação fechada de acordo com a própria semântica do recurso, enquanto a representação fechada em uma solicitação PUT é definida como substituindo o estado do recurso de destino. Portanto, a intenção de PUT é idempotente e visível para intermediários, mesmo que o efeito exato seja conhecido apenas pelo servidor de origem.

A interpretação adequada de uma solicitação PUT pressupõe que o agente do usuário saiba qual recurso de destino é desejado. Um serviço que seleciona um URI adequado em nome do cliente, depois de receber uma solicitação de alteração de estado, DEVE ser implementado usando o método POST em vez de PUT. Se o servidor de origem não fizer a alteração de estado PUT solicitada no recurso de destino e, em vez disso, desejar que ela seja aplicada a um recurso diferente, como quando o recurso foi movido para um URI diferente, o servidor de origem DEVERÁ enviar uma resposta 3xx (Redirecionamento) apropriada; o agente do usuário PODE então tomar sua própria decisão sobre redirecionar ou não a solicitação.

Uma solicitação PUT aplicada ao recurso de destino pode ter efeitos colaterais em outros recursos. Por exemplo, um artigo pode ter um URI para identificar "a versão atual" (um recurso) que é separado dos

URIs que identificam cada versão específica (recursos diferentes

que em um ponto compartilhou o mesmo estado que o recurso da versão atual). Uma solicitação PUT bem-sucedida no URI "da versão atual" pode, portanto, criar um novo recurso de versão, além de alterar o estado do recurso de destino, e também pode fazer com que links sejam adicionados entre os recursos relacionados.

Um servidor de origem que permite PUT em um determinado recurso de destino DEVE enviar uma resposta 400 (Solicitação Incorreta) a uma solicitação PUT que contenha um Content-Range (Seção 4.2 de [RFC7233]), já que a carga provavelmente será um conteúdo parcial que foi erroneamente PUT como uma representação completa. As atualizações parciais de conteúdo são possíveis direcionando um recurso identificado separadamente com estado que se sobrepõe a uma parte do recurso maior ou usando um método diferente que foi definido especificamente para atualizações parciais (por exemplo, o método PATCH definido em [RFC5789]).

As respostas ao método PUT não são armazenável em cache. Se uma solicitação PUT bem-sucedida passar por um cache que tenha uma ou mais respostas armazenadas para o URI de solicitação efetivo, essas respostas armazenadas serão invalidadas (consulte a Seção 4.4 de [RFC7234]).

4.3.5. EXCLUIR

O método DELETE solicita que o servidor de origem remova a associação entre o recurso de destino e seu funcionalidade. Na verdade, esse método é semelhante ao comando rm no UNIX: ele expressa uma operação de exclusão no mapeamento de URI do servidor de origem, em vez de uma expectativa de que as informações associadas anteriormente sejam excluídas.

Se o recurso de destino tiver uma ou mais representações atuais, elas poderão ou não ser destruídas pelo servidor de origem, e o armazenamento associado poderá ou não ser recuperado, dependendo inteiramente da natureza do recurso e de sua implementação pelo servidor de origem (que estão além do escopo desta especificação). Da mesma forma, outros aspectos de implementação de um recurso podem precisar ser desativados ou arquivados como resultado de um DELETE, como banco de dados ou gateway Conexões. Em geral, supõe-se que o servidor de origem só permitirá DELETE em recursos para os quais ele tenha um mecanismo prescrito para realizar a exclusão.

Relativamente poucos recursos permitem o método DELETE -- seu uso principal é para ambientes de criação remota, onde o usuário tem alguma orientação sobre seu efeito. Por exemplo, um recurso que foi criado anteriormente usando uma solicitação PUT ou identificado por meio do campo de cabeçalho Local após uma resposta 201 (Criada) a uma solicitação POST pode permitir que uma solicitação DELETE correspondente desfaça essas Ações. Da mesma forma, implementações de agente de usuário personalizadas que implementam

uma função de criação, como clientes de controle de revisão usando HTTP para operações remotas, pode usar DELETE com base na suposição de que o espaço de URI do servidor foi criado para corresponder a um repositório de versão.

Se um método DELETE for aplicado com êxito, o servidor de origem DEVERÁ enviar um código de status 202 (Aceito) se a ação provavelmente for bem-sucedida, mas ainda não tiver sido promulgada, um código de status 204 (Sem Conteúdo) se a ação tiver sido promulgada e nenhuma informação adicional for fornecida, ou um código de status 200 (OK) se a ação tiver sido promulgada e a mensagem de resposta incluir uma representação descrevendo o status.

Uma carga útil dentro de uma mensagem de solicitação DELETE não tem semântica definida; enviar um corpo de carga útil em uma solicitação DELETE pode fazer com que algumas implementações existentes rejeitem a solicitação.

As respostas ao método DELETE não podem ser armazenadas em cache. Se uma solicitação DELETE passar por um cache que tenha uma ou mais respostas armazenadas para o URI de solicitação efetivo, essas respostas armazenadas serão invalidadas (consulte a Seção 4.4 de [RFC7234]).

4.3.6. LIGAR

O método CONNECT solicita que o destinatário estabeleça um túnel para o servidor de origem de destino identificado pelo destino da solicitação e, se for bem-sucedido, restrinja seu comportamento ao encaminhamento cego de pacotes, em ambas as direções, até que o túnel seja fechado. Os túneis são comumente usados para criar uma conexão virtual de ponta a ponta, por meio de um ou mais proxies, que podem ser protegidos usando TLS (Transport Layer Security, [RFC5246]).

CONNECT destina-se apenas ao uso em solicitações a um procuração. Um servidor de origem que recebe uma solicitação CONNECT para si mesmo PODE responder com um código de status 2xx (bem-sucedido) para indicar que uma conexão é
estabelecido. No entanto, a maioria dos servidores de origem não implementa CONNECT.

Um cliente que envia uma solicitação CONNECT DEVE enviar o formulário de autoridade de solicitação-alvo (Seção 5.3 de [RFC7230]); ou seja, o destino da solicitação consiste apenas no nome do host e no número da porta do destino do túnel, separados por dois pontos. Por exemplo

```
CONECTE-SE server.example.com:80 host
HTTP/1.1: server.example.com:80
```

O proxy do destinatário pode estabelecer um túnel conectando-se

diretamente ao destino da solicitação ou, se configurado para usar outro proxy, encaminhando a solicitação CONNECT para o próximo proxy de entrada. Qualquer resposta 2xx (bem-sucedida) indica que o remetente (e todos os

proxies de entrada) mudará para o modo de túnel imediatamente após a linha em branco que conclui a seção de cabeçalho da resposta bem-sucedida; Os dados recebidos após essa linha em branco são do servidor identificado pelo destino da solicitação. Qualquer resposta que não seja uma resposta bem-sucedida indica que o túnel ainda não foi formado e que a conexão permanece regida por HTTP.

Um túnel é fechado quando um intermediário de túnel detecta que um dos lados fechou sua conexão: o intermediário DEVE tentar enviar todos os dados pendentes que vieram do lado fechado para o outro lado, fechar ambas as conexões e, em seguida, descartar todos os dados restantes não entregues.

A autenticação de proxy pode ser usada para estabelecer a autoridade para criar um túnel. Por exemplo

```
CONECTE-SE server.example.com:80 host
HTTP/1.1: server.example.com:80
Proxy-Authorization: basic aGVsbG86d29ybGQ=
```

Há riscos significativos no estabelecimento de um túnel para servidores arbitrários, especialmente quando o destino é uma porta TCP bem conhecida ou reservada que não se destina ao tráfego da Web. Por exemplo, um CONNECT para um destino de solicitação de "example.com:25" sugeriria que o proxy se conectasse à porta reservada para tráfego SMTP; Se permitido, isso pode induzir o proxy a retransmitir e-mails de spam. Os proxies que suportam CONNECT DEVEM restringir seu uso a um conjunto limitado de portas conhecidas ou a uma lista de permissões configurável de destinos de solicitação segura.

Um servidor NÃO DEVE enviar nenhum campo de cabeçalho Transfer-Encoding ou Content-Length em uma resposta 2xx (bem-sucedida) para CONNECT. Um cliente DEVE ignorar todos os campos de cabeçalho Content-Length ou Transfer-Encoding recebidos em uma resposta bem-sucedida a CONNECT.

Uma carga útil dentro de uma mensagem de solicitação CONNECT não tem semântica definida; enviar um corpo de carga útil em uma solicitação CONNECT pode fazer com que algumas implementações existentes rejeitem a solicitação.

As respostas ao método CONNECT não podem ser armazenadas em cache.

4.3.7. OPÇÕES

O método OPTIONS solicita informações sobre as opções de comunicação disponíveis para o recurso de destino, no servidor de origem ou em um intermediário. Esse método permite que um cliente determine as opções e/ou requisitos associados a um recurso ou os recursos de um servidor, sem implicar em uma ação de recurso.

Uma solicitação OPTIONS com um asterisco ("*") como alvo da solicitação (Seção 5.3 de [RFC7230]) se aplica ao servidor em geral, e não a um específico recurso. Como as opções de comunicação de um servidor normalmente dependem do recurso, a solicitação "*" só é útil como um tipo de método "ping" ou "no-op"; ele não faz nada além de permitir que o cliente teste os recursos do servidor. Por exemplo, isso pode ser usado para testar um proxy para conformidade com HTTP/1.1 (ou falta dela).

Se o destino da solicitação não for um asterisco, a solicitação OPTIONS se aplicará às opções disponíveis ao se comunicar com o recurso de destino.

Um servidor que gera uma resposta bem-sucedida a OPTIONS DEVE enviar quaisquer campos de cabeçalho que possam indicar recursos opcionais implementados pelo servidor e aplicáveis ao recurso de destino (por exemplo, Permitir), incluindo extensões potenciais não definidas por esta especificação.
A carga útil da resposta, se houver, também pode descrever as opções de comunicação em uma máquina ou legível por humanos representação. Um formato padrão para tal representação não é definido por essa especificação, mas pode ser definido por futuras extensões para HTTP. Um servidor DEVE gerar um campo Content-Length com um valor de "0" se nenhum corpo de carga útil for enviado na resposta.

Um cliente PODE enviar um campo de cabeçalho Max-Forwards em uma solicitação OPTIONS para direcionar um destinatário específico na cadeia de solicitações (consulte
Seção 5.1.2). Um proxy NÃO DEVE gerar um campo de cabeçalho Max-Forwards ao encaminhar uma solicitação, a menos que essa solicitação tenha sido recebida com um campo Max-Forwards.

Um cliente que gera uma solicitação OPTIONS contendo um corpo de carga DEVE enviar um campo de cabeçalho Content-Type válido descrevendo o tipo de mídia de representação. Embora essa especificação não defina nenhum uso para essa carga, extensões futuras para HTTP podem usar o corpo OPTIONS para fazer consultas mais detalhadas sobre o recurso de destino.

As respostas ao método OPTIONS não podem ser armazenadas em cache.

4.3.8. TRAÇO

O método TRACE solicita um loopback remoto no nível do aplicativo da solicitação Mensagem. O destinatário final da solicitação DEVE refletir a mensagem recebida, excluindo alguns campos descritos abaixo, de volta ao cliente como o corpo da mensagem de uma resposta 200 (OK) com um Content-Type de "message/http" (Seção 8.3.1 de [RFC7230]). O destinatário final é o servidor de origem ou o primeiro servidor a receber um valor Max-Forwards de zero (0) na solicitação

(Seção 5.1.2).

Um cliente NÃO DEVE gerar campos de cabeçalho em uma solicitação TRACE contendo dados confidenciais que podem ser divulgados pela resposta. Por exemplo, seria tolice para um agente de usuário enviar credenciais de usuário armazenadas [RFC7235] ou cookies [RFC6265] em um TRACE pedir. O destinatário final da solicitação DEVE excluir todos os campos de cabeçalho da solicitação que provavelmente conterão dados confidenciais quando esse destinatário gerar o corpo da resposta.

O TRACE permite que o cliente veja o que está sendo recebido na outra extremidade da cadeia de solicitações e use esses dados para teste ou diagnóstico informação. O valor do campo de cabeçalho Via (Seção 5.7.1 de [RFC7230]) é de particular interesse, pois atua como um rastreamento da solicitação cadeia. O uso do campo de cabeçalho Max-Forwards permite que o cliente limite o comprimento da cadeia de solicitações, o que é útil para testar uma cadeia de proxies encaminhando mensagens em um loop infinito.

Um cliente NÃO DEVE enviar um corpo de mensagem em uma

solicitação TRACE. As respostas ao método TRACE não podem

ser armazenadas em cache.

5. Campos de cabeçalho de solicitação

Um cliente envia campos de cabeçalho de solicitação para fornecer mais informações sobre o contexto da solicitação, tornar a solicitação condicional com base no estado do recurso de destino, sugerir formatos preferenciais para a resposta, fornecer credenciais de autenticação ou modificar o processamento de solicitação esperado. Esses campos atuam como modificadores de solicitação, semelhantes aos parâmetros em uma invocação de método de linguagem de programação.

5.1. Controles

Os controles são campos de cabeçalho de solicitação que direcionam o tratamento específico da solicitação.

Nome do campo de cabeçalho	Definido em...
Controle de cache	Secção 5.2 de [RFC7234]
Esperar	Secção 5.1.1
Anfitrião	Secção 5.4 de [RFC7230]
Max-Forwards	Secção 5.1.2
Pragma	Secção 5.4 de [RFC7234]
Gama	Secção 3.1 de [RFC7233]
TE	Secção 4.3 de [RFC7230]

5.1.1. Esperar

O campo de cabeçalho "Esperar" em uma solicitação indica um determinado conjunto de comportamentos (expectativas) que precisam ser suportados pelo servidor para lidar adequadamente com isso pedir. A única expectativa definida por essa especificação é 100-continue.

```
Expect= "100-continue"
```

O valor do campo Expect não diferencia maiúsculas de minúsculas.

Um servidor que recebe um valor de campo Esperado diferente de 100-continue PODE responder com um código de status 417 (Falha na Expectativa) para indicar que a expectativa inesperada não pode ser atendida.

Uma expectativa de 100 continues informa aos destinatários que o cliente está prestes a enviar um corpo de mensagem (presumivelmente grande) nessa solicitação e deseja receber uma resposta provisória de 100 (Continue) se o
Os campos de linha de solicitação e cabeçalho não são suficientes para causar uma resposta imediata de sucesso, redirecionamento ou erro. Isso permite que o cliente aguarde uma indicação de que vale a pena enviar o corpo da mensagem antes de realmente fazê-lo, o que pode melhorar a eficiência quando o corpo da mensagem é enorme ou quando o cliente antecipa que um erro é provável (por exemplo, ao enviar um método de alteração de estado, pela primeira vez, sem credenciais de autenticação verificadas anteriormente).

Por exemplo, uma solicitação que começa

```
com PUT /somewhere/fun HTTP/1.1
Host: origin.example.com Tipo
de conteúdo: vídeo / h264
Duração do conteúdo:
1234567890987 Esperar: 100-
continuar
```

permite que o servidor de origem responda imediatamente com uma mensagem de erro, como 401 (Não autorizado) ou 405 (Método não permitido), antes que o cliente comece a preencher os pipes com uma transferência de dados desnecessária.

Requisitos para clientes:

- Um cliente NÃO DEVE gerar uma expectativa de 100 continues em uma solicitação que não inclui um corpo de mensagem.

- Um cliente que aguardará uma resposta 100 (Continue) antes de

enviar o corpo da mensagem de solicitação DEVE enviar um campo de cabeçalho Expect contendo uma expectativa de 100-continue.

- Um cliente que envia uma expectativa de 100 continuações não precisa aguardar nenhum período de tempo específico; esse cliente PODE continuar a enviar o corpo da mensagem mesmo que ainda não tenha recebido uma resposta. Além disso, como 100 respostas (Continue) não podem ser enviadas por meio de um intermediário HTTP/1.0, esse cliente NÃO DEVE esperar por um período indefinido antes de enviar o corpo da mensagem.

- Um cliente que recebe um código de status 417 (Falha na expectativa) em resposta a uma solicitação que contém uma expectativa de 100 continuações DEVE repetir essa solicitação sem uma expectativa de 100 continuações, pois o
  A resposta 417 apenas indica que a cadeia de resposta não suporta expectativas (por exemplo, ela passa por um servidor HTTP/1.0).

Requisitos para servidores:

- Um servidor que recebe uma expectativa de 100 continues em uma solicitação HTTP/1.0 DEVE ignorar essa expectativa.

- Um servidor PODE omitir o envio de uma resposta 100 (Continuar) se já tiver recebido parte ou todo o corpo da mensagem para a solicitação correspondente ou se o enquadramento indicar que não há corpo da mensagem.

- Um servidor que envia uma resposta 100 (Continuar) DEVE, em última análise, enviar um código de status final, uma vez que o corpo da mensagem é recebido e processado, a menos que a conexão seja fechada prematuramente.

- Um servidor que responde com um código de status final antes de ler todo o corpo da mensagem DEVE indicar nessa resposta se pretende fechar a conexão ou continuar lendo e descartando a mensagem de solicitação (consulte a Seção 6.6 de [RFC7230]).

Um servidor de origem DEVE, ao receber uma linha de solicitação HTTP/1.1 (ou posterior) e uma seção de cabeçalho completa que contenha um
100-continue e indique que um corpo de mensagem de solicitação se seguirá, envie uma resposta imediata com um código de status final, se esse status puder ser determinado examinando apenas os campos de linha de solicitação e cabeçalho, ou envie uma resposta imediata 100 (Continue) para incentivar o cliente a enviar a mensagem da solicitação corpo. O servidor de origem NÃO DEVE aguardar o corpo da mensagem antes de enviar a resposta 100 (Continuar).

Um proxy DEVE, ao receber uma linha de solicitação HTTP/1.1 (ou posterior) e uma seção de cabeçalho completa que contenha uma expectativa de 100 continues e indique que um corpo de mensagem de solicitação se seguirá, enviar uma resposta imediata com um código de status final, se esse status puder ser determinado examinando

apenas os campos de linha de solicitação e cabeçalho, ou começar a encaminhar a solicitação para o servidor de origem enviando um

seção de linha de solicitação e cabeçalho correspondente para o próximo servidor de entrada. Se o proxy acreditar (a partir da configuração ou interação anterior) que o próximo servidor de entrada suporta apenas HTTP/1.0, o proxy PODE gerar uma resposta imediata 100 (Continuar) para incentivar o cliente a começar a enviar o corpo da mensagem.

> Observação: o campo de cabeçalho Expect foi adicionado após a publicação original do HTTP/1.1 [RFC2068] como o meio de solicitar uma resposta 100 (Continue) provisória e o mecanismo geral para indicar extensões de compreensão obrigatória. No entanto, o mecanismo de extensão não foi usado por clientes e os requisitos obrigatórios não foram implementados por muitos servidores, tornando o mecanismo de extensão inútil. Essa especificação removeu o mecanismo de extensão para simplificar a definição e o processamento de 100-continue.

5.1.2. Max-Forwards

O campo de cabeçalho "Max-Forwards" fornece um mecanismo com os métodos de solicitação TRACE (Seção 4.3.8) e OPTIONS (Seção 4.3.7) para limitar o número de vezes que a solicitação é encaminhada por proxies. Isso pode ser útil quando o cliente está tentando rastrear uma solicitação que parece estar falhando ou fazendo um loop no meio da cadeia.

```
Max-Forwards = 1*DÍGITO
```

O valor Max-Forwards é um número inteiro decimal que indica o número restante de vezes que essa mensagem de solicitação pode ser encaminhada.

Cada intermediário que recebe uma solicitação TRACE ou OPTIONS contendo um campo de cabeçalho Max-Forwards DEVE verificar e atualizar seu valor antes de encaminhar a solicitação. Se o valor recebido for zero (0), o intermediário NÃO DEVE encaminhar a solicitação; em vez disso, o intermediário DEVE responder como o destinatário final. Se o valor de Max-Forwards recebido for maior que zero, o intermediário DEVE gerar um campo Max-Forwards atualizado na mensagem encaminhada com um valor de campo que seja o menor de a) o valor recebido diminuído por um (1) ou b) o valor máximo suportado do destinatário para Max-Forwards.

Um destinatário PODE ignorar um campo de cabeçalho Max-Forwards recebido com qualquer outro método de solicitação.

5.2. Condicionais

Os campos de cabeçalho de solicitação condicional HTTP [RFC7232] permitem que um cliente coloque uma pré-condição no estado do recurso de destino, de modo que a ação correspondente à semântica do método

não seja aplicada se a pré-condição for avaliada como falsa. Cada condição prévia definida por

esta especificação consiste em uma comparação entre um conjunto de validadores obtidos a partir de representações anteriores do recurso de destino com o estado atual dos validadores para a representação selecionada (Seção 7.2). Portanto, essas pré-condições avaliam se o estado do recurso de destino foi alterado desde um determinado estado conhecido pelo cliente. O efeito de tal avaliação depende da semântica do método e da escolha da condicional, conforme definido na Seção 5 de [RFC7232].

Campo de cabeçalho Nome	Definido em...
Correspondência	Secção 3.1 de [RFC7232]
Se-Nenhuma-Correspondência	Secção 3.2 de [RFC7232]
Se-Modificado-Desde	Secção 3.3 de [RFC7232]
Se-Não modificado-Desde	Secção 3.4 de [RFC7232]
Intervalo Se	Secção 3.2 de [RFC7233]

5.3. Negociação de conteúdo

Os campos de cabeçalho de solicitação a seguir são enviados por um agente de usuário para se envolver na negociação proativa do conteúdo da resposta, conforme definido na Seção 3.4.1. As preferências enviadas nesses campos se aplicam a qualquer conteúdo na resposta, incluindo representações do recurso de destino, representações de erro ou status de processamento e, potencialmente, até mesmo as cadeias de caracteres de texto diversas que podem aparecer no protocolo.

Nome do campo de cabeçalho	Definido em...
Aceitar	Seção 5.3.2
Aceitar-Charset	Seção 5.3.3
Aceitar-Codificação	Secção 5.3.4
Aceitar-Idioma	Seção 5.3.5

5.3.1. Valores de qualidade

Muitos dos campos de cabeçalho de solicitação para negociação proativa usam um parâmetro comum, chamado "q" (sem distinção entre maiúsculas e minúsculas), para atribuir um "peso" relativo à preferência por esse tipo de conteúdo associado. Esse peso é chamado de "valor de qualidade" (ou "qvalue") porque o mesmo nome de parâmetro é frequentemente usado nas configurações do servidor para atribuir um peso à qualidade relativa das várias representações que podem ser selecionadas para um recurso.

O peso é normalizado para um número real no intervalo de 0 a 1, em que 0,001 é o menos preferido e 1 é o mais preferido; Um valor de 0 significa "não aceitável". Se nenhum parâmetro "q" estiver presente, o peso padrão será 1.

```
peso = OWS ";" OWS "q=" qvalue
qvalue = ( "0" [ "." 0*3
DÍGITOS])
       / ( "1" [ "." 0*3("0") ] )
```

Um remetente de qvalue NÃO DEVE gerar mais de três dígitos após o decimal ponto. A configuração do usuário desses valores deve ser limitada da mesma maneira.

5.3.2. Aceitar

O campo de cabeçalho "Aceitar" pode ser usado por agentes de usuário para especificar tipos de mídia de resposta aceitáveis. Os campos de cabeçalho Accept podem ser usados para indicar que a solicitação é especificamente limitada a um pequeno conjunto de tipos desejados, como no caso de uma solicitação para um
imagem em linha.

```
Aceitar = #( intervalo de mídia [ accept-

params ] ) intervalo de mídia= ( "*/*"
                / ( digite "/" "*" )
                / ( tipo "/" subtipo )
                ) *( OWS ";" Parâmetro OWS )
accept-params = weight *( accept-ext )
aceitar-ext = OWS ";" Token OWS [ "=" ( token / string-aspas ) ]
```

O caractere asterisco "*" é usado para agrupar tipos de mídia em intervalos, com "*/*" indicando todos os tipos de mídia e "type/*" indicando todos os subtipos desse tipo. O intervalo de mídia pode incluir parâmetros de tipo de mídia aplicáveis a esse intervalo.

Cada intervalo de mídia pode ser seguido por zero ou mais parâmetros de tipo de mídia aplicáveis (por exemplo, conjunto de caracteres), um parâmetro opcional "q" para indicar um peso relativo (Seção 5.3.1) e, em seguida, zero ou mais parâmetros de extensão. O parâmetro "q" é necessário se alguma extensão (accept-ext) estiver presente, pois atua como um separador entre os dois conjuntos de parâmetros.

> Nota: O uso do nome do parâmetro "q" para separar os parâmetros do tipo de mídia dos parâmetros de extensão Accept é devido ao histórico prática. Embora isso impeça que qualquer parâmetro de tipo de mídia chamado "q" seja usado com um intervalo de mídia, acredita-se que tal evento seja improvável, dada a falta de parâmetros "q" na IANA

registro de tipo de mídia e o uso raro de qualquer parâmetro de tipo de mídia em Os tipos de mídia Accept.Future não são recomendados a registrar qualquer parâmetro chamado "q".

O exemplo

Aceitar: áudio/*; q=0,2, áudio/básico

é interpretado como "Eu prefiro áudio/básico, mas me envie qualquer tipo de áudio se for o melhor disponível após uma redução de 80% na qualidade".

Uma solicitação sem nenhum campo de cabeçalho Accept implica que o agente do usuário aceitará qualquer tipo de mídia em resposta. Se o campo de cabeçalho estiver presente em uma solicitação e nenhuma das representações disponíveis para a resposta tiver um tipo de mídia listado como aceitável, o servidor de origem poderá honrar o campo de cabeçalho enviando uma resposta 406 (Não Aceitável) ou desconsiderar o campo de cabeçalho tratando a resposta como se ela não estivesse sujeita à negociação de conteúdo.

Um exemplo mais elaborado é

```
Aceitar: texto/simples; q=0,5,
        texto/html, texto/x-dvi;
        q=0,8, texto/x-c
```

Verbalmente, isso seria interpretado como "text/html e text/x-c são os tipos de mídia igualmente preferidos, mas se eles não existirem, envie a representação text/x-dvi e, se isso não existir, envie a representação text/plain".

Os intervalos de mídia podem ser substituídos por intervalos de mídia mais específicos ou tipos de mídia específicos. Se mais de um intervalo de mídia se aplicar a um determinado tipo, a referência mais específica terá precedência. Por exemplo

```
Aceitar: texto/*, texto/simples, texto/simples;
```

format=flowed, */* têm a seguinte precedência:

1. texto/simples; formato = fluiu
2. texto/simples
3. Texto/*
4. */*

O fator de qualidade do tipo de mídia associado a um determinado tipo é determinado localizando o intervalo de mídia com a

precedência mais alta que corresponde ao tipo. Por exemplo

```
  Aceitar: texto/*; q=0,3, texto/html; q=0,7, texto/html; nível = 1,
          texto / html; nível = 2; q=0,4, */*; q=0,5
```

faria com que os seguintes valores fossem associados:

Mídia Tipo	Valor da Qualidade
texto/html; nível = 1	1
texto/html	0,7
texto/simples	0,3
imagem/jpeg	0.5
texto/html; nível = 2	0.4
texto/html; nível = 3	0,7

Nota: Um agente de usuário pode ser fornecido com um conjunto padrão de valores de qualidade para determinadas mídias Intervalos. No entanto, a menos que o agente do usuário seja um sistema fechado que não pode interagir com outros agentes de renderização, esse conjunto padrão deve ser configurável pelo usuário.

5.3.3. Aceitar-Charset

O campo de cabeçalho "Accept-Charset" pode ser enviado por um agente do usuário para indicar quais conjuntos de caracteres são aceitáveis no conteúdo da resposta textual. Esse campo permite que agentes de usuário capazes de entender conjuntos de caracteres mais abrangentes ou de propósito especial sinalizem esse recurso para um servidor de origem capaz de representar informações nesses conjuntos de caracteres.

```
  Aceitar-Charset = 1#( ( charset / "*" ) [ peso ] )
```

Os nomes dos conjuntos de caracteres são definidos na Seção 3.1.1.2. Um agente de usuário PODE associar um valor de qualidade a cada conjunto de caracteres para indicar a preferência relativa do usuário por esse conjunto de caracteres, conforme definido na Seção 5.3.1. Um exemplo é

```
  Aceitar-Charset: iso-8859-5, unicode-1-1; q=0,8
```

O valor especial "*", se presente no campo Accept-Charset, corresponde a todos os conjuntos de caracteres que não são mencionados em outro lugar no campo Accept-Charset. Se nenhum "*" estiver presente em um campo Accept-Charset, todos os conjuntos de caracteres não mencionados explicitamente no campo serão considerados "não aceitáveis" para o cliente.

Uma solicitação sem nenhum campo de cabeçalho Accept-Charset implica que o agente do usuário aceitará qualquer conjunto de caracteres em

resposta. A maioria dos agentes de usuário de uso geral não envia
Accept-Charset, a menos que especificamente

configurado para fazer isso, porque uma lista detalhada de conjuntos de caracteres suportados torna mais fácil para um servidor identificar um indivíduo em virtude das características de solicitação do agente do usuário (Seção 9.7).

Se um campo de cabeçalho Accept-Charset estiver presente em uma solicitação e nenhuma das representações disponíveis para a resposta tiver um conjunto de caracteres listado como aceitável, o servidor de origem poderá honrar o campo de cabeçalho, enviando uma resposta 406 (Não Aceitável), ou desconsiderar o campo de cabeçalho tratando o recurso como se ele não estivesse sujeito à negociação de conteúdo.

5.3.4. Aceitar Codificação

O campo de cabeçalho "Accept-Encoding" pode ser usado por agentes de usuário para indicar quais codificações de conteúdo de resposta (Seção 3.1.2.1) são aceitáveis na resposta. Um token de "identidade" é usado como sinônimo de "sem codificação" para se comunicar quando nenhuma codificação é preferida.

```
  Aceitar-Codificação = #( codificações [ peso ] )
  codings= codificação de conteúdo / "identidade" / "*"
```

Cada valor de codificação PODE receber um valor de qualidade associado que representa a preferência por essa codificação, conforme definido na Seção 5.3.1.O símbolo de asterisco "*" num campo Accept-Encoding corresponde a qualquer codificação de conteúdo disponível não explicitamente listada no campo de cabeçalho.

Por exemplo

```
  Aceitar-Codificação: compactar,
  gzip Aceitar-Codificação:
  Aceitar-Codificação: *
  Aceitar-Codificação: compactar; q=0,5, gzip; q=1,0
  Aceitar-Codificação: gzip; q=1,0, identidade; q=0,5, *; q=0
```

Uma solicitação sem um campo de cabeçalho Accept-Encoding implica que o agente do usuário não tem preferências em relação a codificações de conteúdo. Embora isso permita que o servidor use qualquer codificação de conteúdo em uma resposta, isso não implica que o agente do usuário será capaz de processar corretamente todas as codificações.

Um servidor testa se uma codificação de conteúdo para uma determinada representação é aceitável usando estas regras:

1. Se nenhum campo Accept-Encoding estiver na solicitação, qualquer codificação de conteúdo será considerada aceitável pelo agente do usuário.

2. Se a representação não tiver codificação de conteúdo, ela será aceitável por padrão, a menos que seja especificamente excluída pelo campo Accept-Encoding informando "identidade; q=0" ou "*; q=0" sem uma entrada mais específica para "identidade".

3. Se a codificação de conteúdo da representação for uma das codificações de conteúdo listadas no campo Accept-Encoding, ela será aceitável, a menos que seja acompanhada por um valor q de 0. (Conforme definido na Seção 5.3.1, um valor q de 0 significa "não aceitável".)

4. Se várias codificações de conteúdo forem aceitáveis, a codificação de conteúdo aceitável com o valor q mais alto diferente de zero será preferida.

Um campo de cabeçalho Accept-Encoding com um valor de campo combinado vazio implica que o agente do usuário não deseja nenhuma codificação de conteúdo em resposta. Se um campo de cabeçalho Accept-Encoding estiver presente em uma solicitação e nenhuma das representações disponíveis para a resposta tiver uma codificação de conteúdo listada como aceitável, o servidor de origem DEVERÁ enviar uma resposta sem qualquer codificação de conteúdo.

> Nota: A maioria dos aplicativos HTTP/1.0 não reconhece ou obedece a qvalues associados a codificações de conteúdo. Isso significa que qvalues podem não funcionar e não são permitidos com x-gzip ou x-compress.

5.3.5. Linguagem de aceitação

O campo de cabeçalho "Accept-Language" pode ser usado por agentes de usuário para indicar o conjunto de idiomas naturais preferidos na resposta. As tags de idioma são definidas na Seção 3.1.3.1.

```
  Aceitar-Linguagem = 1#( intervalo-de-idioma [
  peso ] ) intervalo de idioma =
            <intervalo de idiomas, consulte [RFC4647], Seção 2.1>
```

Cada intervalo de idiomas pode receber um valor de qualidade associado que representa uma estimativa da preferência do usuário pelos idiomas especificados por esse intervalo, conforme definido na Seção 5.3.1. Por exemplo

```
  Aceitar-Idioma: da, en-gb; q=0,8, pol; q=0,7
```

significaria: "Prefiro dinamarquês, mas aceitarei o inglês britânico e outros tipos de inglês".

Uma solicitação sem nenhum campo de cabeçalho Accept-Language implica que o agente do usuário aceitará qualquer idioma em resposta. Se o

campo de cabeçalho estiver presente em uma solicitação e nenhuma das representações disponíveis para a resposta tiver uma marca de idioma correspondente, o servidor de origem poderá desconsiderar o campo de cabeçalho tratando a resposta como se

não está sujeito a negociação de conteúdo ou respeita o campo de cabeçalho enviando um 406 (Não Aceitável) resposta. No entanto, este último não é incentivado, pois isso pode impedir que os usuários acessem conteúdo que possam usar (com software de tradução, por exemplo).

Observe que alguns destinatários tratam a ordem na qual as tags de idioma são listadas como uma indicação de prioridade decrescente, especialmente para tags que recebem valores de qualidade iguais (nenhum valor é o mesmo que q=1). No entanto, esse comportamento não pode ser confiável. Para consistência e para maximizar a interoperabilidade, muitos agentes de usuário atribuem a cada tag de idioma um valor de qualidade exclusivo, ao mesmo tempo em que os listam em ordem decrescente qualidade. Discussões adicionais sobre listas de prioridades linguísticas podem ser encontradas na Seção 2.3 de [RFC4647].

Para correspondência, a Seção 3 de [RFC4647] define várias correspondências Esquemas. As implementações podem oferecer o esquema de correspondência mais apropriado para seus Requisitos. O esquema de "Filtragem Básica" ([RFC4647], Seção 3.3.1) é idêntico ao esquema de correspondência que foi definido anteriormente para HTTP na Seção 14.4 de [RFC2616].

Pode ser contrário às expectativas de privacidade do usuário enviar um campo de cabeçalho Accept-Language com as preferências linguísticas completas do usuário em cada solicitação (Seção 9.7).

Como a inteligibilidade é altamente dependente do usuário individual, os agentes do usuário precisam permitir o controle do usuário sobre a preferência linguística (seja por meio da configuração do próprio agente do usuário ou por padrão para um sistema controlável pelo usuário configuração). Um agente de usuário que não fornece esse controle ao usuário NÃO DEVE enviar um
Campo de cabeçalho Accept-Language.

> Observação: os agentes do usuário devem fornecer orientação aos usuários ao definir uma preferência, pois os usuários raramente estão familiarizados com os detalhes da correspondência de idioma, conforme descrito acima. Por exemplo, os usuários podem presumir que, ao selecionar "en-gb", eles receberão qualquer tipo de documento em inglês se o inglês britânico não for disponível. Um agente de usuário pode sugerir, nesse caso, adicionar "en" à lista para um melhor comportamento de correspondência.

5.4. Credenciais de autenticação

Dois campos de cabeçalho são usados para transportar credenciais de autenticação, conforme definido em [RFC7235]. Observe que vários mecanismos personalizados para autenticação do usuário usam o campo de cabeçalho Cookie para essa finalidade, conforme definido em [RFC6265].

Campo de cabeçalho Nome	Definido em...
Autorização	Secção 4.2 de [RFC7235]
Autorização de proxy	Secção 4.4 de [RFC7235]

5.5. Contexto da solicitação

Os campos de cabeçalho de solicitação a seguir fornecem informações adicionais sobre o contexto da solicitação, incluindo informações sobre o usuário, o agente do usuário e o recurso por trás da solicitação.

Nome do campo de cabeçalho	Definido em...
A partir de	Seção 5.5.1
Referência	Seção 5.5.2
Agente do usuário	Seção 5.5.3

5.5.1. De

O campo de cabeçalho "De" contém um endereço de e-mail da Internet para um usuário humano que controla o usuário solicitante agente. O endereço deve ser utilizável por máquina, conforme definido por "caixa de correio" na Seção 3.4 de [RFC5322]:

> De= caixa de correio
>
> mailbox = <mailbox, consulte [RFC5322], Seção

3.4> Um exemplo é:

> De: webmaster@example.org

O campo de cabeçalho De raramente é enviado por um usuário não robótico Agentes. Um agente de usuário NÃO DEVE enviar um campo de cabeçalho From sem configuração explícita pelo usuário, pois isso pode entrar em conflito com os interesses de privacidade do usuário ou com a política de segurança de seu site.

Um agente de usuário robótico DEVE enviar um campo de cabeçalho From válido para que a pessoa responsável por executar o robô possa ser contatada se ocorrerem problemas nos servidores, como se o robô estiver enviando solicitações excessivas, indesejadas ou inválidas.

Um servidor NÃO DEVE usar o campo de cabeçalho De para controle de acesso ou autenticação, pois a maioria dos destinatários assumirá que o valor do campo é uma informação pública.

5.5.2. Referenciador

O campo de cabeçalho "Referer" [sic] permite que o agente do usuário especifique uma referência de URI para o recurso do qual o URI de destino foi obtido (ou seja, o "referenciador", embora o nome do campo esteja incorreto). Um agente de usuário NÃO DEVE incluir os componentes fragment e userinfo da referência de URI [RFC3986], se houver, ao gerar o valor do campo Referer.

```
Referer = URIs-absolutos/URIs-parciais
```

O campo de cabeçalho Referer permite que os servidores gerem backlinks para outros recursos para análises simples, registro, cache otimizado, etc. Ele também permite que links obsoletos ou digitados incorretamente sejam encontrados para manutenção. Alguns servidores usam o campo de cabeçalho Referer como um meio de negar links de outros sites (o chamado "deep linking") ou restringir a falsificação de solicitação entre sites (CSRF), mas nem todas as solicitações o contêm.

Exemplo:

```
Referência: http://www.example.org/hypertext/Overview.html
```

Se o URI de destino foi obtido de uma fonte que não tem seu próprio URI (por exemplo, entrada do teclado do usuário ou uma entrada nos favoritos/favoritos do usuário), o agente do usuário DEVE excluir o campo Referer ou enviá-lo com um valor de "about:blank".

O campo Referer tem o potencial de revelar informações sobre o contexto da solicitação ou o histórico de navegação do usuário, o que é uma preocupação de privacidade se o identificador do recurso de referência revelar informações pessoais (como um nome de conta) ou um recurso que deveria ser confidencial (como atrás de um firewall ou interno de um serviço seguro). A maioria dos agentes de usuário de uso geral não envia o campo de cabeçalho Referer quando o recurso de referência é um URI local de "arquivo" ou "dados". Um agente de usuário NÃO DEVE enviar um campo de cabeçalho Referer em uma solicitação HTTP não segura se a página de referência foi recebida com um protocolo. Consulte a Seção 9.4 para considerações adicionais de segurança.

Sabe-se que alguns intermediários removem indiscriminadamente os campos de cabeçalho Referer das solicitações de saída. Isso tem o infeliz efeito colateral de interferir na proteção contra ataques CSRF, que podem ser muito mais prejudiciais aos usuários. Intermediários e extensões de agente de usuário que desejam limitar a divulgação de informações no Referer devem restringir suas alterações a edições específicas, como substituir nomes de domínio internos por pseudônimos ou truncar a consulta e/ou o caminho Componentes. Um intermediário NÃO DEVE modificar ou excluir o campo de cabeçalho Referer quando o valor do campo compartilha o mesmo esquema e host que o destino da solicitação.

5.5.3. Agente do usuário

O campo de cabeçalho "User-Agent" contém informações sobre o agente do usuário que originou a solicitação, que geralmente é usado pelos servidores para ajudar a identificar o escopo dos problemas de interoperabilidade relatados, para contornar ou adaptar respostas para evitar limitações específicas do agente do usuário e para análises relacionadas ao navegador ou sistema operacional usar. Um agente de usuário DEVE enviar um campo User-Agent em cada solicitação, a menos que especificamente configurado para não fazê-lo.

```
User-Agent = produto *( RWS ( produto / comentário ) )
```

O valor do campo User-Agent consiste em um ou mais identificadores de produto, cada um seguido por zero ou mais comentários (Seção 3.2 de [RFC7230]), que juntos identificam o software do agente do usuário e seus subprodutos. Por convenção, os identificadores de produto são listados em ordem decrescente de importância para identificar o agente do usuário software. Cada identificador de produto consiste em um nome e uma versão opcional.

```
product= token ["/" versão-do-produto]
versão-do-produto = token
```

Um remetente DEVE limitar os identificadores de produto gerados ao que é necessário para identificar o produto; um remetente NÃO DEVE gerar publicidade ou outras informações não essenciais dentro do identificador do produto. Um remetente NÃO DEVE gerar informações em versão do produto que não é um identificador de versão (ou seja, versões sucessivas do mesmo nome de produto devem diferir apenas na parte da versão do produto do identificador do produto).

Exemplo:

```
Agente do usuário: CERN-LineMode / 2.15 libwww / 2.17b3
```

Um agente de usuário NÃO DEVE gerar um campo User-Agent contendo detalhes desnecessariamente refinados e DEVE limitar a adição de subprodutos por terceiros. Valores de campo User-Agent excessivamente longos e detalhados aumentam a latência da solicitação e o risco de um usuário ser identificado contra sua vontade ("impressão digital").

Da mesma forma, as implementações são encorajadas a não usar os tokens de produto de outras implementações para declarar compatibilidade com eles, pois isso contorna o propósito do campo. Se um agente de usuário se disfarçar como um agente de usuário diferente, os destinatários podem presumir que o usuário deseja intencionalmente ver respostas personalizadas para esse agente de usuário identificado, mesmo que elas não funcionem tão bem para o agente de usuário real que está sendo usado.

6. Códigos de status de resposta

O elemento status-code é um código inteiro de três dígitos que fornece o resultado da tentativa de entender e satisfazer a solicitação.

Os códigos de status HTTP são extensíveis. Os clientes HTTP não são obrigados a entender o significado de todos os códigos de status registrados, embora tal compreensão seja obviamente desejável. No entanto, um cliente DEVE entender a classe de qualquer código de status, conforme indicado pelo primeiro dígito, e tratar um código de status não reconhecido como equivalente ao código de status x00 dessa classe, com a exceção de que um destinatário NÃO DEVE armazenar em cache uma resposta com um código de status não reconhecido.

Por exemplo, se um código de status não reconhecido de 471 for recebido por um cliente, o cliente poderá presumir que houve algo errado com sua solicitação e tratar a resposta como se tivesse recebido um código de status 400 (Solicitação Incorreta). A mensagem de resposta geralmente contém uma representação que explica o status.

O primeiro dígito do código de status define a classe de resposta. Os dois últimos dígitos não têm nenhuma categorização papel. Existem cinco valores para o primeiro dígito:

- 1xx (Informativo): A solicitação foi recebida, processo contínuo
- 2xx (bem-sucedida): a solicitação foi recebida, compreendida e aceita com sucesso
- 3xx (Redirecionamento): Outras ações precisam ser tomadas para concluir a solicitação
- 4xx (Erro do cliente): A solicitação contém sintaxe incorreta ou não pode ser atendida

- 5xx (Erro do servidor): O servidor falhou ao atender a uma solicitação aparentemente válida

6.1. Visão geral dos códigos de status

Os códigos de status listados abaixo são definidos nesta especificação, na Seção 4 de [RFC7232], na Seção 4 de [RFC7233] e na Seção 3 de [RFC7235]. As frases de motivo listadas aqui são apenas recomendações
-- eles podem ser substituídos por equivalentes locais sem afetar o protocolo.

Respostas com códigos de status definidos como armazenáveis em cache por padrão (por exemplo, 200, 203, 204, 206, 300, 301, 404, 405, 410, 414 e 501 em
esta especificação) pode ser reutilizada por um cache com expiração heurística, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos [RFC7234]; Todos os outros códigos de status não podem ser armazenados em cache por padrão.

Código	Frase-Razão	Definido em...
100	Continuar	Secção 6.2.1
101	Protocolos de comutação	Secção 6.2.2
200	OKEY	Secção 6.3.1
201	Criado	Secção 6.3.2
202	Aceitado	Secção 6.3.3
203	Informações não autorizadas	Secção 6.3.4
204	Sem conteúdo	Secção 6.3.5
205	Redefinir conteúdo	Secção 6.3.6
206	Conteúdo parcial	Secção 4.1 de [RFC7233]
300	Múltiplas escolhas	Secção 6.4.1
301	Movido permanentemente	Secção 6.4.2
302	Fundar	Secção 6.4.3
303	Ver outros	Secção 6.4.4
304	Não modificado	Secção 4.1 de [RFC7232]
305	Usar proxy	Secção 6.4.5
307	Redirecionamento temporário	Secção 6.4.7
400	Solicitação incorreta	Secção 6.5.1
401	Desautorizado	Secção 3.1 de [RFC7235]
402	Pagamento Obrigatório	Secção 6.5.2
403	Proibido	Secção 6.5.3
404	Não encontrado	Secção 6.5.4
405	Método não permitido	Secção 6.5.5
406	Não aceitável	Secção 6.5.6
407	Autenticação de proxy necessária	Secção 3.2 de [RFC7235]
408	Tempo limite da solicitação	Secção 6.5.7
409	Conflito	Secção 6.5.8
410	Sumido	Secção 6.5.9
411	Comprimento necessário	Secção 6.5.10
412	Falha na pré-condição	Secção 4.2 de [RFC7232]
413	Carga útil muito grande	Secção 6.5.11
414	URI muito longo	Secção 6.5.12
415	Tipo de mídia não suportado	Secção 6.5.13
416	Intervalo não satisfatível	Secção 4.4 de [RFC7233]
417	Falha na expectativa	Secção 6.5.14
426	Atualização necessária	Secção 6.5.15
500	Erro interno do servidor	Secção 6.6.1
501	Não implementado	Secção 6.6.2
502	Bad Gateway	Secção 6.6.3
503	Serviço indisponível	Secção 6.6.4
504	Tempo limite do gateway	Secção 6.6.5
505	Versão HTTP não suportada	Secção 6.6.6

Observe que esta lista não é exaustiva - ela não inclui códigos de status de extensão definidos em outras especificações. A lista completa de códigos de status é mantida pela IANA. Consulte a Seção 8.2 para obter detalhes.

6.2. Informativo 1xx

A classe 1xx (Informativa) do código de status indica uma resposta provisória para comunicar o status da conexão ou o progresso da solicitação antes de concluir a ação solicitada e enviar uma resposta final. As respostas 1xx são encerradas pela primeira linha vazia após a linha de status (a linha vazia sinalizando o fim da seção de cabeçalho). Como o HTTP/1.0 não definiu nenhum código de status 1xx, um servidor NÃO DEVE enviar uma resposta 1xx para um cliente HTTP/1.0.

Um cliente DEVE ser capaz de analisar uma ou mais respostas 1xx recebidas antes de uma resposta final, mesmo que o cliente não espere Um. Um agente de usuário PODE ignorar respostas 1xx inesperadas.

Um proxy DEVE encaminhar respostas 1xx, a menos que o próprio proxy tenha solicitado a geração da resposta 1xx. Por exemplo, se um proxy adicionar um campo "Esperar: 100-continue" ao encaminhar uma solicitação, ele não precisará encaminhar as 100 respostas (Continue) correspondentes.

6.2.1. 100 Continuar

O código de status 100 (Continuar) indica que a parte inicial de uma solicitação foi recebida e ainda não foi rejeitada pelo servidor. O servidor pretende enviar uma resposta final depois que a solicitação tiver sido totalmente recebida e atendida.

Quando a solicitação contém um campo de cabeçalho Expect que inclui um 100-continue, a resposta 100 indica que o servidor deseja receber o corpo da carga útil da solicitação, conforme descrito em Secção 5.1.1. O cliente deve continuar enviando a solicitação e descartar a resposta 100.

Se a solicitação não contiver um campo de cabeçalho Expect contendo a expectativa de 100 continues, o cliente poderá simplesmente descartar essa resposta provisória.

6.2.2. 101 Protocolos de comutação

O código de status 101 (Switching Protocols) indica que o servidor entende e está disposto a atender à solicitação do cliente, por meio do campo Upgrade header (Seção 6.7 de [RFC7230]), para uma alteração no protocolo de aplicativo que está sendo usado nesta conexão. O servidor

DEVE gerar um campo de cabeçalho Upgrade na resposta que indica para quais protocolos serão alternados imediatamente após a linha vazia que encerra a resposta 101.

Supõe-se que o servidor só concordará em trocar de protocolo quando for vantajoso fazê-lo. Por exemplo, mudar para uma versão mais recente do HTTP pode ser vantajoso em relação às versões mais antigas, e mudar para um protocolo síncrono em tempo real pode ser vantajoso ao fornecer recursos que usam esses recursos.

6.3. 2xx de sucesso

A classe 2xx (Bem-sucedida) do código de status indica que a solicitação do cliente foi recebida, compreendida e aceita com êxito.

6.3.1. 200 OK

O código de status 200 (OK) indica que a solicitação foi bem-sucedida. A carga útil enviada em uma resposta 200 depende do método de solicitação. Para os métodos definidos por essa especificação, o significado pretendido da carga útil pode ser resumido como:

GET uma representação do recurso de destino;

HEADa mesma representação que GET, mas sem os dados de representação;

POSTuma representação do status ou dos resultados obtidos da ação;

PUT, DELETE uma representação do status da ação; OPÇÕES uma

representação das opções de comunicação;

TRACE uma representação da mensagem de solicitação conforme recebida pelo servidor final.

Além das respostas a CONNECT, uma resposta 200 sempre tem uma carga útil, embora um servidor de origem POSSA gerar um corpo de carga útil de comprimento zero.
Se nenhuma carga for desejada, um servidor de origem deverá enviar 204 (Sem Conteúdo). Para CONNECT, nenhuma carga é permitida porque o resultado bem-sucedido é um túnel, que começa imediatamente após a seção do cabeçalho de resposta 200.

Uma resposta 200 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.3.2. 201 Criado

O código de status 201 (Criado) indica que a solicitação foi atendida e resultou em um ou mais novos recursos sendo Criado. O recurso primário criado pela solicitação é identificado por um campo de cabeçalho Local na resposta ou, se nenhum campo Local for recebido, pelo URI de solicitação efetivo.

A carga útil da resposta 201 normalmente descreve e vincula aos recursos criados. Consulte a Seção 7.2 para obter uma discussão sobre o significado e a finalidade dos campos de cabeçalho do validador, como ETag e
Última modificação, em uma resposta 201.

6.3.3. 202 Aceitos

O código de status 202 (Aceito) indica que a solicitação foi aceita para processamento, mas o processamento não foi concluído. A solicitação pode ou não ser atendida, pois pode não ser permitida quando o processamento realmente ocorre. Não há nenhum recurso no HTTP para reenviar um código de status de uma operação assíncrona.

A resposta 202 é intencionalmente descompromissada. Sua finalidade é permitir que um servidor aceite uma solicitação para algum outro processo (talvez um processo orientado a lotes que é executado apenas uma vez por dia) sem exigir que a conexão do agente do usuário com o servidor persista até que o processo seja concluído. A representação enviada com essa resposta deve descrever o status atual da solicitação e apontar para (ou inserir) um monitor de status que possa fornecer ao usuário uma estimativa de quando a solicitação será atendida.

6.3.4. 203 Informações não autorizadas

O código de status 203 (Informações não autoritativas) indica que a solicitação foi bem-sucedida, mas a carga anexa foi modificada da resposta 200 (OK) do servidor de origem por um proxy de transformação (Seção 5.7.2 de [RFC7230]). Esse código de status permite que o proxy notifique os destinatários quando uma transformação for aplicada, pois esse conhecimento pode afetar decisões posteriores sobre o conteúdo. Por exemplo, futuras solicitações de validação de cache para o conteúdo podem ser aplicáveis apenas ao longo do mesmo caminho de solicitação (por meio dos mesmos proxies).

A resposta 203 é semelhante ao código de aviso de 214 Transformação aplicada (Seção 5.5 de [RFC7234]), que tem a vantagem de ser aplicável a respostas com qualquer código de status.

Uma resposta 203 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.3.5. 204 Sem conteúdo

O código de status 204 (Sem Conteúdo) indica que o servidor atendeu com êxito à solicitação e que não há conteúdo adicional a ser enviado no corpo da carga útil da resposta. Os metadados nos campos de cabeçalho de resposta referem-se ao recurso de destino e sua representação selecionada após a aplicação da ação solicitada.

Por exemplo, se um código de status 204 for recebido em resposta a uma solicitação PUT e a resposta contiver um campo de cabeçalho ETag, o PUT foi bem-sucedido e o valor do campo ETag conterá a marca de entidade para a nova representação desse recurso de destino.

A resposta 204 permite que um servidor indique que a ação foi aplicada com êxito ao recurso de destino, ao mesmo tempo em que implica que o agente do usuário não precisa sair de sua "exibição de documento" atual (se houver). O servidor assume que o agente do usuário fornecerá alguma indicação do sucesso ao seu usuário, de acordo com sua própria interface, e aplicará quaisquer metadados novos ou atualizados na resposta à sua representação ativa.

Por exemplo, um código de status 204 é comumente usado com interfaces de edição de documentos correspondentes a uma ação "salvar", de modo que o documento que está sendo salvo permaneça disponível para o usuário por edição. Também é frequentemente usado com interfaces que esperam que as transferências de dados automatizadas sejam predominantes, como em sistemas de controle de versão distribuídos.

Uma resposta 204 é encerrada pela primeira linha vazia após os campos de cabeçalho porque não pode conter um corpo de mensagem.

Uma resposta 204 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.3.6. 205 Redefinir conteúdo

O código de status 205 (Redefinir conteúdo) indica que o servidor atendeu à solicitação e deseja que o agente do usuário redefina a "visualização do documento", que fez com que a solicitação fosse enviada, para seu estado original, conforme recebido do servidor de origem.

Essa resposta destina-se a dar suporte a um caso de uso comum de entrada de dados em que o usuário recebe conteúdo que dá suporte à entrada de dados (um formulário, bloco de notas, tela etc.), insere ou manipula dados nesse espaço,

faz com que os dados inseridos sejam enviados em uma solicitação e, em seguida, o mecanismo de entrada de dados é redefinido para a próxima entrada para que o usuário possa iniciar facilmente outra ação de entrada.

Como o código de status 205 implica que nenhum conteúdo adicional será fornecido, um servidor NÃO DEVE gerar uma carga útil em uma resposta 205. Em outras palavras, um servidor DEVE fazer um dos seguintes para uma resposta 205: a) indicar um corpo de comprimento zero para a resposta, incluindo um campo de cabeçalho Content-Length com um valor de 0; b) indicar uma carga útil de comprimento zero para a resposta, incluindo um campo de cabeçalho Transfer-Encoding com um valor de chunked e um corpo de mensagem que consiste em um único chunk de comprimento zero; ou, c) feche a conexão imediatamente após o envio da linha em branco que termina a seção do cabeçalho.

6.4. Redirecionamento 3xx

A classe 3xx (Redirection) do código de status indica que outras ações precisam ser tomadas pelo agente do usuário para cumprir o pedir. Se um campo de cabeçalho Location (Seção 7.1.2) for fornecido, o agente do usuário PODERÁ redirecionar automaticamente sua solicitação para o URI referenciado pelo valor do campo Location, mesmo que o código de status específico não seja compreendido. O redirecionamento automático precisa ser feito com cuidado para métodos não conhecidos como seguros, conforme definido na Seção 4.2.1, uma vez que o usuário pode não querer redirecionar uma solicitação não segura.

Existem vários tipos de redirecionamentos:

1. Redirecionamentos que indicam que o recurso pode estar disponível em um URI diferente, conforme fornecido pelo campo Local, como nos códigos de status 301 (Movido Permanentemente), 302 (Encontrado) e 307 (Redirecionamento Temporário).

2. Redirecionamento que oferece uma opção de recursos correspondentes, cada um capaz de representar o destino da solicitação original, como no Código de status 300 (Múltiplas Escolhas).

3. Redirecionamento para um recurso diferente, identificado pelo campo Local, que pode representar uma resposta indireta à solicitação, como no código de status 303 (Consulte Outro).

4. Redirecionamento para um resultado armazenado em cache anteriormente, como no código de status 304 (Não Modificado).

   Observação: no HTTP/1.0, os códigos de status 301 (movido permanentemente) e

302 (Encontrado) foram definidos para o primeiro tipo de redirecionamento ([RFC1945], Seção 9.3). Os primeiros agentes de usuário se dividiram sobre se o método aplicado ao destino de redirecionamento seria o mesmo que o

solicitação original ou seria reescrito como GET. Embora o HTTP tenha definido originalmente a semântica anterior para 301 e 302 (para corresponder à sua implementação original no CERN) e definido 303 (ver outro) para corresponder à última semântica, a prática predominante gradualmente convergiu para a última semântica para 301 e 302 também. A primeira revisão do HTTP/1.1 adicionou 307 (Redirecionamento Temporário) para indicar a semântica anterior sem ser afetada por divergências prática. Mais de 10 anos depois, a maioria dos agentes de usuário ainda reescreve o método para 301 e 302; portanto, essa especificação torna esse comportamento compatível quando a solicitação original é POST.

Um cliente DEVE detectar e intervir em redirecionamentos cíclicos (ou seja, loops de redirecionamento "infinitos").

Nota: Uma versão anterior desta especificação recomendava um máximo de cinco redirecionamentos ([RFC2068], Seção 10.3). Os desenvolvedores de conteúdo precisam estar cientes de que alguns clientes podem implementar essa limitação fixa.

6.4.1. 300 Escolhas Múltiplas

O código de status 300 (Multiple Choices) indica que o recurso de destino tem mais de uma representação, cada uma com seu próprio identificador mais específico, e informações sobre as alternativas estão sendo fornecidas para que o usuário (ou agente do usuário) possa selecionar uma representação preferencial redirecionando sua solicitação para uma ou mais delas Identificadores. Em outras palavras, o servidor deseja que o agente do usuário se envolva em negociação reativa para selecionar a(s) representação(ões) mais apropriada(s) para suas necessidades (Seção 3.4).

Se o servidor tiver uma escolha preferencial, o servidor DEVE gerar um campo de cabeçalho Local contendo a referência de URI de uma escolha preferencial. O agente do usuário PODE usar o valor do campo Local para redirecionamento automático.

Para métodos de solicitação diferentes de HEAD, o servidor DEVE gerar uma carga útil na resposta 300 contendo uma lista de metadados de representação e referência(s) de URI a partir da qual o usuário ou agente de usuário pode escolher o mais preferido. O agente do usuário PODE fazer uma seleção dessa lista automaticamente se entender o tipo de mídia fornecido. Um formato específico para seleção automática não é definido por essa especificação porque o HTTP tenta permanecer ortogonal à definição de suas cargas. Na prática, a representação é fornecida em algum formato facilmente analisado que se acredita ser aceitável para o agente do usuário, conforme determinado pelo design compartilhado ou negociação de conteúdo, ou em algum formato de hipertexto comumente aceito.

Uma resposta 300 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

> Nota: A proposta original para o código de status 300 definiu o campo de cabeçalho URI como fornecendo uma lista de representações alternativas, de modo que seria utilizável para 200, 300 e
> 406 respostas e ser transferido em respostas ao método HEAD. No entanto, a falta de implantação e desacordo sobre a sintaxe levaram ao URI e às alternativas (uma proposta subsequente) a serem retirados desta especificação. É possível comunicar a lista usando um conjunto de campos de cabeçalho Link [RFC5988], cada um com uma relação de "alternado", embora a implantação seja um problema do ovo e da galinha.

6.4.2. 301 movido permanentemente

O código de status 301 (Movido Permanentemente) indica que o recurso de destino recebeu um novo URI permanente e quaisquer referências futuras a esse recurso devem usar um dos URIs incluídos. Os clientes com recursos de edição de link devem vincular automaticamente as referências ao URI de solicitação efetivo a uma ou mais das novas referências enviadas pelo servidor, sempre que possível.

O servidor DEVE gerar um campo de cabeçalho Location na resposta contendo uma referência de URI preferencial para o novo URI permanente. O agente do usuário PODE usar o valor do campo Local para redirecionamento. A carga de resposta do servidor geralmente contém uma pequena nota de hipertexto com um hiperlink para o(s) novo(s) URI(s).

> Observação: por motivos históricos, um agente de usuário PODE alterar o método de solicitação de POST para GET para o subsequente pedir. Se esse comportamento for indesejado, o código de status 307 (Redirecionamento Temporário) poderá ser usado.

Uma resposta 301 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.4.3. 302 Encontrado

O código de status 302 (Encontrado) indica que o recurso de destino reside temporariamente em um URI. Como o redirecionamento pode ser alterado ocasionalmente, o cliente deve continuar a usar o URI de solicitação efetivo para solicitações futuras.

O servidor DEVE gerar um campo de cabeçalho Location na resposta contendo uma referência de URI para o URI diferente. O agente do usuário PODE usar o valor do campo Local para redirecionamento automático. A carga de resposta do servidor geralmente contém uma pequena nota de hipertexto com um hiperlink para os diferentes URIs.

> Observação: por motivos históricos, um agente de usuário PODE alterar o método de solicitação de POST para GET para o subsequente pedir. Se esse comportamento for indesejado, o código de status 307 (Redirecionamento Temporário) poderá ser usado.

6.4.4. 303 Ver outros

O código de status 303 (Ver Outro) indica que o servidor está redirecionando o agente do usuário para um recurso diferente, conforme indicado por um URI no campo de cabeçalho Local, que se destina a fornecer uma resposta indireta ao original pedir. Um agente de usuário pode executar uma solicitação de recuperação direcionada a esse URI (uma solicitação GET ou HEAD se estiver usando HTTP), que também pode ser redirecionada e apresentar o resultado final como uma resposta à solicitação original. Observe que o novo URI no campo de cabeçalho Local não é considerado equivalente ao URI de solicitação efetivo.

Este código de status é aplicável a qualquer HTTP método. Ele é usado principalmente para permitir a saída de uma ação POST para redirecionar o agente do usuário para um recurso selecionado, pois isso fornece as informações correspondentes à resposta POST em um formato que pode ser identificado, marcado e armazenado em cache separadamente, independentemente da solicitação original.

Uma resposta 303 a uma solicitação GET indica que o servidor de origem não tem uma representação do recurso de destino que pode ser transferido pelo servidor por HTTP. No entanto, o valor do campo Local refere-se a um recurso que é descritivo do recurso de destino, de modo que fazer uma solicitação de recuperação nesse outro recurso pode resultar em uma representação útil para os destinatários sem implicar que ele representa o recurso de destino original. Observe que as respostas às perguntas sobre o que pode ser representado, quais representações são adequadas e o que pode ser uma descrição útil estão fora do escopo do HTTP.

Exceto para respostas a uma solicitação HEAD, a representação de uma resposta 303 deve conter uma breve nota de hipertexto com um hiperlink para a mesma referência de URI fornecida no campo de cabeçalho Local.

6.4.5. 305 Usar proxy

O código de status 305 (Usar Proxy) foi definido em uma versão anterior desta especificação e agora está obsoleto (Apêndice B).

6.4.6. 306 (não utilizado)

O código de status 306 foi definido em uma versão anterior desta especificação, não é mais usado e o código é reservado.

6.4.7. 307 Redirecionamento temporário

O código de status 307 (Redirecionamento Temporário) indica que o recurso de destino reside temporariamente em um URI diferente e o agente do usuário NÃO DEVE alterar o método de solicitação se executar um redirecionamento automático para esse URI. Como o redirecionamento pode mudar ao longo do tempo, o cliente deve continuar usando o URI de solicitação efetivo original para solicitações futuras.

O servidor DEVE gerar um campo de cabeçalho Location na resposta contendo uma referência de URI para o URI diferente. O agente do usuário PODE usar o valor do campo Local para redirecionamento automático. A carga de resposta do servidor geralmente contém uma pequena nota de hipertexto com um hiperlink para os diferentes URIs.

> Observação: esse código de status é semelhante a 302 (Encontrado), exceto que não permite alterar o método de solicitação de POST para GET. Esta especificação não define nenhuma contraparte equivalente para 301 (Movido Permanentemente) ([RFC7238], no entanto, define o código de status 308 (Redirecionamento Permanente) para essa finalidade).

6.5. Erro do cliente 4xx

A classe 4xx (Erro do Cliente) do código de status indica que o cliente parece ter Errou. Exceto ao responder a uma solicitação HEAD, o servidor DEVE enviar uma representação contendo uma explicação da situação de erro e se é uma condição temporária ou permanente. Esses códigos de status são aplicáveis a qualquer método de solicitação. Os agentes do usuário DEVEM exibir qualquer representação incluída para o usuário.

6.5.1. 400 Solicitação Incorreta

O código de status 400 (Solicitação Incorreta) indica que o servidor não pode ou não processará a solicitação devido a algo que é percebido como um erro do cliente (por exemplo, sintaxe de solicitação malformada, enquadramento de mensagem de solicitação inválido ou roteamento de solicitação enganoso).

6.5.2. 402 Pagamento necessário

O código de status 402 (Pagamento Obrigatório) é reservado para uso futuro.

6.5.3. 403 Proibido

O código de status 403 (Proibido) indica que o servidor entendeu a solicitação, mas se recusa a autorizar ela. Um servidor que deseja tornar público por que a solicitação foi proibida pode descrever esse motivo na carga útil da resposta (se houver).

Se as credenciais de autenticação foram fornecidas na solicitação, o servidor as considerará insuficientes para conceder acesso. O cliente NÃO DEVE repetir automaticamente a solicitação com o mesmo credenciais. O cliente PODE repetir a solicitação com novo ou diferente credenciais. No entanto, uma solicitação pode ser proibida por motivos não relacionados às credenciais.

Um servidor de origem que deseja "ocultar" a existência atual de um recurso de destino proibido PODE, em vez disso, responder com um código de status de
404 (não encontrado).

6.5.4. 404 Não encontrado

O código de status 404 (Não Encontrado) indica que o servidor de origem não encontrou uma representação atual para o recurso de destino ou não está disposto a divulgá-la Existe. Um código de status 404 não indica se essa falta de representação é temporária ou permanente; o código de status 410 (Gone) é preferível a 404 se o servidor de origem souber, presumivelmente por algum meio configurável, que a condição provavelmente será permanente.

Uma resposta 404 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.5.5. 405 Método não permitido

O código de status 405 (Método Não Permitido) indica que o método recebido na linha de solicitação é conhecido pelo servidor de origem, mas não é compatível com o recurso de destino. O servidor de origem DEVE gerar um campo de cabeçalho Allow em uma resposta 405 contendo uma lista dos métodos atualmente suportados pelo recurso de destino.

Uma resposta 405 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.5.6. 406 Não aceitável

O código de status 406 (Não Aceitável) indica que o recurso de destino não tem uma representação atual que seria aceitável para o agente do usuário, de acordo com os campos de cabeçalho de negociação proativa recebidos na solicitação (Seção 5.3), e o servidor não está disposto a fornecer uma representação padrão.

O servidor DEVE gerar uma carga contendo uma lista de características de representação disponíveis e identificadores de recursos correspondentes dos quais o usuário ou agente de usuário pode escolher o mais apropriado. Um agente de usuário PODE selecionar automaticamente a opção mais apropriada dessa lista. No entanto, esta especificação não define nenhum padrão para essa seleção automática, conforme descrito na Seção 6.4.1.

6.5.7. 408 Tempo limite da solicitação

O código de status 408 (Tempo limite da solicitação) indica que o servidor não recebeu uma mensagem de solicitação completa dentro do tempo em que estava preparado para aguardar. Um servidor DEVE enviar a opção de conexão "fechar" (Seção 6.1 de [RFC7230]) na resposta, uma vez que 408 implica que o servidor decidiu fechar a conexão em vez de continuar espera. Se o cliente tiver uma solicitação pendente em trânsito, o cliente PODERÁ repetir essa solicitação em uma nova conexão.

6.5.8. 409 Conflito

O código de status 409 (Conflito) indica que a solicitação não pôde ser concluída devido a um conflito com o estado atual do recurso de destino. Esse código é usado em situações em que o usuário pode resolver o conflito e reenviar o pedir. O servidor DEVE gerar uma carga útil que inclua informações suficientes para que um usuário reconheça a origem do conflito.

É mais provável que ocorram conflitos em resposta a um PUT pedir. Por exemplo, se o controle de versão estivesse sendo usado e a representação que está sendo PUT incluísse alterações em um recurso que entrassem em conflito com as feitas por uma solicitação anterior (de terceiros), o servidor de origem poderia usar uma resposta 409 para indicar que não pode concluir o pedir. Nesse caso, a representação de resposta provavelmente conteria informações úteis para mesclar as diferenças com base no histórico de revisões.

6.5.9. 410 se foi

O código de status 410 (Gone) indica que o acesso ao recurso de destino não está mais disponível no servidor de origem e que essa condição provavelmente será permanente. Se o servidor de origem não

saber, ou não tem facilidade para determinar, se a condição é permanente ou não, o código de status 404 (Não encontrado) deve ser usado em seu lugar.

A resposta 410 destina-se principalmente a auxiliar a tarefa de manutenção da web, notificando o destinatário de que o recurso está intencionalmente indisponível e que os proprietários do servidor desejam que os links remotos para esse recurso sejam removidos. Esse evento é comum para serviços promocionais por tempo limitado e para recursos pertencentes a indivíduos que não estão mais associados ao site do servidor de origem. Não é necessário marcar todos os recursos permanentemente indisponíveis como "perdidos" ou manter a marca por qualquer período de tempo - isso fica a critério do proprietário do servidor.

Uma resposta 410 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.5.10. 411 Comprimento necessário

O código de status 411 (Length Required) indica que o servidor se recusa a aceitar a solicitação sem um Content-Length definido (Seção 3.3.2 de [RFC7230]). O cliente PODE repetir a solicitação se adicionar um campo de cabeçalho Content-Length válido contendo o comprimento do corpo da mensagem na mensagem de solicitação.

6.5.11. 413 Carga útil muito grande

O código de status 413 (Payload Too Large) indica que o servidor está se recusando a processar uma solicitação porque a carga útil da solicitação é maior do que o servidor deseja ou é capaz de processo. O servidor PODE fechar a conexão para impedir que o cliente continue a solicitação.

Se a condição for temporária, o servidor DEVE gerar um Retry-After para indicar que é temporário e após que horas o cliente PODE tentar novamente.

6.5.12. 414 URI muito longo

O código de status 414 (URI Too Long) indica que o servidor está se recusando a atender à solicitação porque o destino da solicitação (Seção
5.3 de [RFC7230]) é mais longo do que o servidor está disposto a interpretar. Essa condição rara só é provável de ocorrer quando um cliente converteu incorretamente uma solicitação POST em uma solicitação GET com informações de consulta longas, quando o cliente desceu para um "buraco negro" de redirecionamento (por exemplo, um prefixo de URI redirecionado que aponta para um sufixo de si mesmo) ou quando o servidor está sob ataque de um cliente tentando explorar

possíveis falhas de segurança.

Uma resposta 414 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.5.13. 415 Tipo de mídia não suportado

O código de status 415 (Tipo de Mídia Não Suportado) indica que o servidor de origem está se recusando a atender à solicitação porque a carga está em um formato não suportado por esse método no recurso de destino. O problema de formato pode ser devido à solicitação indicada
Content-Type ou Content-Encoding, ou como resultado da inspeção direta dos dados.

6.5.14. 417 Expectativa falhou

O código de status 417 (Falha na expectativa) indica que a expectativa fornecida no campo de cabeçalho Esperado da solicitação (Seção 5.1.1) não pôde ser atendida por pelo menos um dos servidores de entrada.

6.5.15. Atualização 426 necessária

O código de status 426 (Atualização necessária) indica que o servidor se recusa a executar a solicitação usando o protocolo atual, mas pode estar disposto a fazê-lo depois que o cliente atualizar para um protocolo diferente. O servidor DEVE enviar um campo de cabeçalho Upgrade em uma resposta 426 para indicar o(s) protocolo(s) necessário(s) (Seção 6.7 de [RFC7230]).

Exemplo:

```
HTTP/1.1 426 Atualização
necessária Atualização:
HTTP/3.0
Conexão: Atualizar
comprimento do conteúdo:
53 Tipo de conteúdo:
texto / simples

Este serviço requer o uso do protocolo HTTP/3.0.
```

6.6. Erro do servidor 5xx

A classe 5xx (Erro do Servidor) do código de status indica que o servidor está ciente de que errou ou é incapaz de executar o método solicitado. Exceto ao responder a uma solicitação HEAD, o servidor DEVE enviar uma representação contendo uma explicação da situação de erro e se é temporária ou permanente

condição. Um agente de usuário DEVE exibir qualquer representação incluída para o utilizador. Esses códigos de resposta são aplicáveis a qualquer método de solicitação.

6.6.1. 500 Erro interno do servidor

O código de status 500 (Erro Interno do Servidor) indica que o servidor encontrou uma condição inesperada que o impediu de atender à solicitação.

6.6.2. 501 Não implementado

O código de status 501 (Não Implementado) indica que o servidor não suporta a funcionalidade necessária para atender ao pedir. Essa é a resposta apropriada quando o servidor não reconhece o método de solicitação e não é capaz de suportá-lo para nenhum recurso.

Uma resposta 501 pode ser armazenada em cache por padrão; ou seja, a menos que indicado de outra forma pela definição do método ou controles de cache explícitos (consulte a Seção 4.2.2 de [RFC7234]).

6.6.3. 502 Bad Gateway

O código de status 502 (Gateway inválido) indica que o servidor, enquanto atuava como gateway ou proxy, recebeu uma resposta inválida de um servidor de entrada acessado ao tentar atender à solicitação.

6.6.4. 503 Serviço indisponível

O código de status 503 (Serviço Indisponível) indica que o servidor não consegue lidar com a solicitação devido a uma sobrecarga temporária ou manutenção programada, o que provavelmente será aliviado após alguns atraso. O servidor PODE enviar um campo de cabeçalho Retry-After
(Seção 7.1.3) para sugerir um tempo apropriado para o cliente aguardar antes de repetir a solicitação.

> Observação: a existência do código de status 503 não implica que um servidor precise usá-lo quando estiver sobrecarregado. Alguns servidores podem simplesmente recusar a conexão.

6.6.5. 504 Tempo limite do gateway

O código de status 504 (Tempo limite do gateway) indica que o servidor, enquanto atuava como um gateway ou proxy, não recebeu uma resposta oportuna de um servidor upstream que precisava acessar para concluir a solicitação.

6.6.6. 505 Versão HTTP não suportada

O código de status 505 (HTTP Version Not Supported) indica que o servidor não suporta ou se recusa a suportar a versão principal do HTTP que foi usada na solicitação Mensagem. O servidor está indicando que não pode ou não quer concluir a solicitação usando a mesma versão principal do cliente, conforme descrito na Seção 2.6 de [RFC7230], exceto com este erro Mensagem. O servidor DEVE gerar uma representação para a resposta 505 que descreva por que essa versão não é suportada e quais outros protocolos são suportados por esse servidor.

7. Campos de cabeçalho de resposta

Os campos de cabeçalho de resposta permitem que o servidor passe informações adicionais sobre a resposta além do que é colocado no linha de status. Esses campos de cabeçalho fornecem informações sobre o servidor, sobre acesso adicional ao recurso de destino ou sobre recursos relacionados.

Embora cada campo de cabeçalho de resposta tenha um significado definido, em geral, a semântica precisa pode ser refinada ainda mais pela semântica do método de solicitação e/ou código de status de resposta.

7.1. Dados de controle

Os campos de cabeçalho de resposta podem fornecer dados de controle que complementam o código de status, direcionam o cache ou instruem o cliente para onde ir em seguida.

Nome do campo de cabeçalho	Definido em...
Idade	Seção 5.1 de [RFC7234]
Controle de cache	Secção 5.2 de [RFC7234]
Expira	Secção 5.3 de [RFC7234]
Encontro	Secção 7.1.1.2
Localização	Secção 7.1.2
Tentar novamente	Secção 7.1.3
Variar	Secção 7.1.4
Atenção	Seção 5.5 de [RFC7234]

7.1.1. Data de Origem

7.1.1.1. Formatos de data/hora

Antes de 1995, havia três formatos diferentes comumente usados pelos servidores para comunicar carimbos de data/hora. Para compatibilidade com implementações antigas, todos os três são definidos aqui. O formato preferencial é um subconjunto de comprimento fixo e de zona única da especificação de data e hora usada pelo Formato de Mensagem da Internet [RFC5322].

HTTP-date= IMF-fixdate / obs-date Um

exemplo do formato preferido é

Dom, 06 de novembro de 1994 08:49:37 GMT; IMF-

fixdate Exemplos dos dois formatos obsoletos são

```
Domingo, 06-Nov-94 08:49:37 GMT; formato RFC 850 obsoleto
Sun Nov 6 08:49:37               1994; Formato asctime() do
ANSI C
```

Um destinatário que analisa um valor de carimbo de data/hora em um campo de cabeçalho HTTP DEVE aceitar todos os três HTTP-date Formatos. Quando um remetente gera um campo de cabeçalho que contém um ou mais carimbos de data/hora definidos como HTTP-date, o remetente DEVE gerar esses carimbos de data/hora no formato IMF-fixdate.

Um valor de data HTTP representa a hora como uma instância do UTC (Tempo Universal Coordenado). Os dois primeiros formatos indicam UTC pela abreviação de três letras para Greenwich Mean Time, "GMT", um predecessor do nome UTC; os valores no formato asctime são considerados em UTC. Um remetente que gera valores de data HTTP a partir de um relógio local deve usar NTP ([RFC5905]) ou algum protocolo semelhante para sincronizar seu relógio com o UTC.

Formato preferido:

```
IMF-fixdate = dia-nome "," SP data1 SP hora do dia SP GMT
; subconjunto fixo de comprimento/zona/capitalização do formato
; ver secção 3.3 de [RFC5322]

nome do      = %x4D.6F.6E ; "Seg",   Diferencia
dia                                  maiúsculas de
                                       minúsculas
             /     %     ; "Ter",   Diferencia
                x54.75.65            maiúsculas de
```

```
                                        minúsculas
/      %      ; "Qua",    Diferencia
   x57.65.64              maiúsculas de
                            minúsculas
/      %      ; "Qui",    Diferencia
   x54.68.75              maiúsculas de
                            minúsculas
/      %      ; "Sex",    Diferencia
   x46.72.69              maiúsculas de
                            minúsculas
/ %x53.61.74 ; "Sento    Diferencia
                 u",     maiúsculas de
                            minúsculas
/ %x53.75.6E ; "Sol",    Diferencia
                         maiúsculas de
                            minúsculas
```

```
     data1        = dia SP mês   Ano SP
                  ; por exemplo, 1982
                    02 Jun

     dia          = 2 DÍGITOS
     mês          = %x4A.61.6E ; "Jan", Diferencia
                                       maiúsculas de
                                       minúsculas
                  / % x46,65,62; "Fev", Diferencia
                  / %x4D.61.72 ; "Mar", maiúsculas  de
                  / %x41.70.72 ; "Abril minúsculasDife
                  / %x4D.61.79 ; ",     rencia
                  / %x4A.75.6E ; "Maio" maiúsculas  de
                  / %x4A.75.6C ; ,      minúsculasDife
                  / %x41.75.67 ; "Jun", rencia
                  / %x53.65.70 ; "Jul", maiúsculas  de
                  / %x4F.63.74 ; "Agost minúsculasDife
                  / %x4E.6F.76 ; o",    renci
                  / % x44,65,63; "Setem
     ano          = 4 DÍGITOS    bro",
                                 "Outub
                                 ro",
                                 "Nov",
                                 "Dez",

     GMT          = %x47.4D.54 ; "GMT", Diferencia
                                       maiúsculas de
                                       minúsculas

     hora do dia = hora ":" minuto ":" segundo
                 ; 00:00:00 - 23:59:60 (segundo bissexto)

     hora= 2 DÍGITOS
     minuto = 2 DÍGITOS
     segundo= 2 dígitos
```

Formatos obsoletos:

```
     obs-data= rfc850-data / asctime-date

     rfc850-data = dia-nome-l "," SP data2 SP hora do dia SP GMT
     data2= dia "-" mês "-" 2DÍGITO
                 ; por exemplo, 02-Jun-82

     nome-do-dia-l=                        %x4D.6F.6E.64.61.79; "Segunda-
     feira", diferencia maiúsculas de minúsculas
            / % x54.75.65.73.64.61.79; "Terça-feira", diferencia maiúsculas
            de minúsculas
            / %x57.65.64.6E.65.73.64.61.79 ; "Quarta-feira", diferencia
            maiúsculas de minúsculas
            / % x54.68.75.72.73.64.61.79; "Quinta-feira", diferencia
            maiúsculas de minúsculas
```

```
        / % x46.72.69.64.61.79; "Sexta-feira", diferencia maiúsculas de
        minúsculas
        / % x53.61.74.75.72.64.61.79; "Sábado", diferencia maiúsculas
        de minúsculas
        / %x53.75.6E.64.61.79; "Domingo", diferencia maiúsculas de
        minúsculas


   asctime-date = dia-nome SP data3 SP hora do dia SP ano
   data3= mês SP ( 2DÍGITO / ( SP 1DÍGITO ))
                ; por exemplo, 2 de junho
```

HTTP-date é maiúsculas e minúsculas sensível. Um remetente NÃO DEVE gerar espaço em branco adicional em uma data HTTP além daquela especificamente incluída como SP no gramática. A semântica de nome do dia, dia, mês, ano e
são as mesmas definidas para as construções de Formato de Mensagem da Internet com o nome correspondente ([RFC5322], Seção 3.3).

Os destinatários de um valor de carimbo de data/hora no formato rfc850-date, que usa um ano de dois dígitos, DEVEM interpretar um carimbo de data/hora que parece estar mais de 50 anos no futuro como representando o ano mais recente no passado que tinha os mesmos dois últimos dígitos.

Os destinatários de valores de carimbo de data/hora são incentivados a serem robustos na análise de carimbos de data/hora, a menos que sejam restritos pela definição de campo. Por exemplo, as mensagens são ocasionalmente encaminhadas por HTTP de um
fonte não HTTP que pode gerar qualquer uma das especificações de data e hora definidas pelo Formato de Mensagem da Internet.

> Observação: os requisitos HTTP para o formato de carimbo de data/hora se aplicam somente ao uso no fluxo de protocolo. As implementações não são necessárias para usar esses formatos para apresentação do usuário, registro de solicitações etc.

7.1.1.2. Data

O campo de cabeçalho "Data" representa a data e a hora em que a mensagem foi originada, tendo a mesma semântica que o Campo de Data de Origem (orig-date) definido na Seção 3.6.1 de [RFC5322]. O valor do campo é uma data HTTP, conforme definido na Seção 7.1.1.1.

```
Data = data HTTP
```

Um exemplo é

```
Data: Terça-feira, 15 de novembro de 1994 08:12:31 GMT
```

Quando um campo de cabeçalho Data é gerado, o remetente DEVE gerar seu valor de campo como a melhor aproximação disponível da data e hora da mensagem geração. Em teoria, a data deve representar o momento imediatamente antes da carga útil ser gerada. Na prática, a data pode ser gerada a qualquer momento durante a origem da mensagem.

Um servidor de origem NÃO DEVE enviar um campo de cabeçalho Date se não tiver um relógio capaz de fornecer uma aproximação razoável da instância atual no Tempo Universal Coordenado. Um servidor de origem PODE enviar um campo de cabeçalho Data se a resposta estiver na classe de códigos de status 1xx (Informativo) ou 5xx (Erro do Servidor). Um servidor de origem DEVE enviar um campo de cabeçalho

Date em todos os outros casos.

Um destinatário com um relógio que recebe uma mensagem de resposta sem um campo de cabeçalho de data DEVE registrar a hora em que foi recebida e anexar um campo de cabeçalho de data correspondente à seção de cabeçalho da mensagem se ela for armazenada em cache ou encaminhada downstream.

Um agente de usuário PODE enviar um campo de cabeçalho Date em uma solicitação, embora geralmente não o faça, a menos que se acredite que transmita informações úteis ao servidor. Por exemplo, aplicativos personalizados de HTTP podem transmitir uma Data se o servidor deve ajustar sua interpretação da solicitação do usuário com base nas diferenças entre o agente do usuário e os relógios do servidor.

7.1.2. Localização

O campo de cabeçalho "Local" é usado em algumas respostas para se referir a um recurso específico em relação ao resposta. O tipo de relacionamento é definido pela combinação de método de solicitação e semântica de código de status.

```
  Localização = referência de URI
```

O valor do campo consiste em uma única referência de URI. Quando ele tem a forma de uma referência relativa ([RFC3986], Seção 4.2), o valor final é calculado resolvendo-o em relação ao URI de solicitação efetivo ([RFC3986], Seção 5).

Para respostas 201 (Created), o valor Location refere-se ao recurso primário criado pelo pedir. Para respostas 3xx (Redirecionamento), o valor Local refere-se ao recurso de destino preferencial para redirecionar automaticamente a solicitação.

Se o valor Location fornecido em uma resposta 3xx (Redirecionamento) não tiver um componente de fragmento, um agente de usuário DEVERÁ processar o redirecionamento como se o valor herdasse o componente de fragmento da referência de URI usada para gerar o destino da solicitação (ou seja, o redirecionamento herda o fragmento da referência original, se houver).

Por exemplo, uma solicitação GET gerada para a referência de URI "http://www.example.org/ t̃im" pode resultar em uma resposta 303 (Ver Outro) contendo o campo de cabeçalho:

```
  Localização: /People.html#tim
```

o que sugere que o agente do usuário
redirecione para
"http://www.example.org/People.html#tim"

Da mesma forma, uma solicitação GET gerada para a referência de URI "http://www.example.org/index.html#larry" pode resultar em uma resposta 301 (Movido Permanentemente) contendo o campo de cabeçalho:

Localização:

http://www.example.net/index.html que sugere

que o agente do usuário redirecione para "http://www.example.net/index.html#larry", preservando o identificador de fragmento original.

Há circunstâncias em que um identificador de fragmento em um valor Location não seria apropriado. Por exemplo, o campo de cabeçalho Location em uma resposta 201 (Created) deve fornecer um URI específico para o recurso criado.

> Observação: Alguns destinatários tentam recuperar de campos de localização que não são URI válidos Referências. Essa especificação não exige ou define esse processamento, mas o permite por uma questão de robustez.
>
> Observação: o campo de cabeçalho Content-Location (Seção 3.1.4.2) difere de Location porque o Content-Location se refere ao recurso mais específico correspondente à representação incluída. Portanto, é possível que uma resposta contenha os campos de cabeçalho Location e Content-Location.

7.1.3. Tentar novamente depois

Os servidores enviam o campo de cabeçalho "Retry-After" para indicar quanto tempo o agente do usuário deve esperar antes de fazer uma solicitação de acompanhamento. Quando enviado com uma resposta 503 (Serviço Indisponível), Retry-After indica por quanto tempo o serviço deve ficar indisponível para o cliente.
Quando enviado com qualquer resposta 3xx (Redirecionamento), Retry-After indica o tempo mínimo que o agente do usuário é solicitado a aguardar antes de emitir a solicitação redirecionada.

O valor desse campo pode ser uma data HTTP ou um número de segundos para atrasar após o recebimento da resposta.

Repetir depois = data HTTP / segundos de atraso

Um valor de atraso de segundos é um inteiro decimal não negativo, representando o tempo em segundos.

segundos de atraso = 1*DÍGITO

Dois exemplos de seu uso são

```
Repetir após: Fri, 31 Dec 1999 23:59:59 GMT
Repetir após: 120
```

No último exemplo, o atraso é de 2 minutos.

7.1.4. Variar

O campo de cabeçalho "Variar" em uma resposta descreve quais partes de uma mensagem de solicitação, além do método, do campo de cabeçalho do host e do destino da solicitação, podem influenciar o processo do servidor de origem para selecionar e representar essa resposta. O valor consiste em um único asterisco ("*") ou em uma lista de nomes de campo de cabeçalho (sem distinção entre maiúsculas e minúsculas).

```
Variar = "*" / 1#nome-do-campo
```

Um valor de campo Vary de "*" sinaliza que qualquer coisa sobre a solicitação pode desempenhar um papel na seleção da representação de resposta, possivelmente incluindo elementos fora da sintaxe da mensagem (por exemplo, o endereço de rede do cliente). Um destinatário não poderá determinar se essa resposta é apropriada para uma solicitação posterior sem encaminhar a solicitação para o servidor de origem. Um proxy NÃO DEVE gerar um campo Vary com um valor "*".

Um valor de campo Vary que consiste em uma lista de nomes separados por vírgulas indica que os campos de cabeçalho de solicitação nomeados, conhecidos como campos de cabeçalho de seleção, podem ter uma função na seleção da representação. Os campos de cabeçalho de seleção potencial não estão limitados aos definidos por esta especificação.

Por exemplo, uma resposta que contém

```
Vary: accept-encoding, accept-language
```

indica que o servidor de origem pode ter usado os campos Accept-Encoding e Accept-Language da solicitação (ou a falta deles) como fatores determinantes ao escolher o conteúdo para essa resposta.

Um servidor de origem pode enviar Vary com uma lista de campos para duas finalidades:

1. Para informar aos destinatários do cache que eles NÃO DEVEM usar essa resposta para atender a uma solicitação posterior, a menos que a solicitação posterior tenha os mesmos valores para os campos listados que a solicitação original (Seção 4.1 de [RFC7234]). Em outras palavras, Vary expande a chave de cache

necessária para corresponder uma nova solicitação à entrada de cache armazenada.

2. Informar aos destinatários do agente do usuário que essa resposta está sujeita à negociação de conteúdo (Seção 5.3) e que uma representação diferente pode ser enviada em uma solicitação subsequente se parâmetros adicionais forem fornecidos nos campos de cabeçalho listados (negociação proativa).

Um servidor de origem DEVE enviar um campo de cabeçalho Vary quando seu algoritmo para selecionar uma representação varia com base em aspectos da mensagem de solicitação diferentes do método e do destino da solicitação, a menos que a variação não possa ser cruzada ou o servidor de origem tenha sido deliberadamente configurado para evitar a transparência do cache. Por exemplo, não há necessidade de enviar o nome do campo Autorização em Variar porque a reutilização entre usuários é restrita pela definição de campo (Seção 4.2 de [RFC7235]). Da mesma forma, um servidor de origem pode usar diretivas Cache-Control (Seção 5.2 de [RFC7234]) para suplantar Vary se considerar a variação menos significativa do que o custo de desempenho do impacto de Vary no cache.

7.2. Campos de cabeçalho do validador

Os campos de cabeçalho do validador transmitem metadados sobre a representação selecionada (Seção 3). Nas respostas a solicitações seguras, os campos do validador descrevem a representação selecionada escolhida pelo servidor de origem ao lidar com a resposta. Observe que, dependendo da semântica do código de status, a representação selecionada para uma determinada resposta não é necessariamente a mesma que a representação incluída como carga útil da resposta.

Em uma resposta bem-sucedida a uma solicitação de alteração de estado, os campos do validador descrevem a nova representação que substituiu a representação selecionada anteriormente como resultado do processamento da solicitação.

Por exemplo, um campo de cabeçalho ETag em uma resposta 201 (Created) comunica a marca de entidade da representação do recurso recém-criado, para que ela possa ser usada em solicitações condicionais posteriores para evitar o problema de "atualização perdida" [RFC7232].

Nome do campo de cabeçalho	Definido em...
ETag	Secção 2.3 de [RFC7232]
Última modificação	Secção 2.2 de [RFC7232]

7.3. Desafios de autenticação

Os desafios de autenticação indicam quais mecanismos estão disponíveis para o cliente fornecer credenciais de autenticação em solicitações futuras.

Nome do campo de cabeçalho	Definido em...
WWW-Autenticar	Seção 4.1 de [RFC7235]
Proxy-Autenticar	Secção 4.3 de [RFC7235]

7.4. Contexto de resposta

Os campos de cabeçalho de resposta restantes fornecem mais informações sobre o recurso de destino para uso potencial em solicitações posteriores.

Nome do campo de cabeçalho	Definido em...
Intervalos de Aceitação	Secção 2.3 de [RFC7233]
Permitir	Secção 7.4.1
Servidor	Secção 7.4.2

7.4.1. Permitir

O campo de cabeçalho "Permitir" lista o conjunto de métodos anunciados como compatíveis com o recurso de destino. A finalidade desse campo é estritamente informar o destinatário sobre métodos de solicitação válidos associados ao recurso.

```
  Permitir =

#method Exemplo

de uso:

  Permitir: OBTER, CABECEAR, COLOCAR
```

O conjunto real de métodos permitidos é definido pelo servidor de origem no momento de cada solicitação. Um servidor de origem DEVE gerar um campo Permitir em uma resposta 405 (Método Não Permitido) e PODE fazê-lo em qualquer outra resposta. Um valor de campo Allow vazio indica que o recurso não permite métodos, o que pode ocorrer em uma resposta 405 se o recurso tiver sido temporariamente desabilitado pela configuração.

Um proxy NÃO DEVE modificar o campo de cabeçalho Allow -- ele não

precisa entender todos os métodos indicados para tratá-los de acordo com as regras genéricas de tratamento de mensagens.

7.4.2. Servidor

O campo de cabeçalho "Servidor" contém informações sobre o software usado pelo servidor de origem para lidar com a solicitação, que geralmente é usado pelos clientes para ajudar a identificar o escopo dos problemas de interoperabilidade relatados, para contornar ou adaptar solicitações para evitar limitações específicas do servidor e para análises relacionadas ao uso do servidor ou do sistema operacional. Um servidor de origem PODE gerar um campo Server em suas respostas.

```
Servidor = produto *( RWS ( produto / comentário ) ) )
```

O valor de campo do servidor consiste em um ou mais identificadores de produto, cada um seguido por zero ou mais comentários (Seção 3.2 de [RFC7230]), que juntos identificam o software do servidor de origem e seu significativo subprodutos. Por convenção, os identificadores de produto são listados em ordem decrescente de importância para identificar o software do servidor de origem. Cada identificador de produto consiste em um nome e uma versão opcional, conforme definido na Seção 5.5.3.

Exemplo:

```
Servidor: CERN/3.0 libwww/2.17
```

Um servidor de origem NÃO DEVE gerar um campo Server contendo detalhes desnecessariamente refinados e DEVE limitar a adição de subprodutos por terceiros. Valores de campo de servidor excessivamente longos e detalhados aumentam a latência de resposta e potencialmente revelam detalhes de implementação interna que podem tornar (um pouco) mais fácil para os invasores encontrar e explorar falhas de segurança conhecidas.

8. Considerações da IANA

8.1. Registro de Método

O "Registro do método HTTP (Hypertext Transfer Protocol)" define o namespace para o token do método de solicitação (Seção 4). O registro do método foi criado e agora é mantido em <http://www.iana.org/assignments/http-methods>.

8.1.1. Procedimento

Os registros de método HTTP DEVEM incluir os seguintes

campos: o Nome do método (consulte a Seção 4)

- Seguro ("sim" ou "não", ver Secção 4.2.1)
- Idempotente ("sim" ou "não", ver ponto 4.2.2)
- Ponteiro para o texto da especificação

Os valores a serem adicionados a este namespace requerem a revisão do IETF (consulte [RFC5226], Seção 4.1).

8.1.2. Considerações para novos métodos

Os métodos padronizados são genéricos; ou seja, eles são potencialmente aplicáveis a qualquer recurso, não apenas a um tipo de mídia específico, tipo de recurso ou aplicação. Como tal, é preferível que novos métodos sejam registrados em um documento que não seja específico para um único aplicativo ou formato de dados, uma vez que as tecnologias ortogonais merecem especificação ortogonal.

Como a análise de mensagens (Seção 3.3 de [RFC7230]) precisa ser independente da semântica do método (além das respostas ao HEAD), as definições de novos métodos não podem alterar o algoritmo de análise ou proibir a presença de um corpo de mensagem na solicitação ou na mensagem de resposta. As definições de novos métodos podem especificar que apenas um corpo de mensagem de comprimento zero é permitido, exigindo um campo de cabeçalho Content-Length com um valor de "0".

Uma nova definição de método precisa indicar se ele é seguro (Seção 4.2.1), idempotente (Seção 4.2.2), armazenável em cache
(Seção 4.2.3), quais semânticas devem ser associadas ao corpo da carga útil, se houver alguma presente na solicitação, e quais refinamentos o método faz no campo de cabeçalho ou na semântica do código de status. Se o novo método for armazenável em cache, sua definição deve descrever como e sob quais condições um cache pode armazenar uma resposta e usá-la para atender a uma solicitação subsequente. O novo método deve descrever se ele pode ser feito condicional (Seção 5.2) e, em caso afirmativo, como um servidor responde quando a condição é falso. Da mesma forma, se o novo método pode ter algum uso para semântica de resposta parcial ([RFC7233]), ele deve documentar isso também.

> Observação: evite definir um nome de método que comece com "M-", pois esse prefixo pode ser mal interpretado como tendo a semântica

atribuída a ele por [RFC2774].

8.1.3. Registros

O "Registro de Método HTTP (Hypertext Transfer Protocol)" foi preenchido com os registros abaixo:

Método	Seguro	Idempotente	Referência
LIGAR	Não	Não	Secção 4.3.6
EXCLUIR	Não	Sim	Secção 4.3.5
OBTER	Sim	Sim	Secção 4.3.1
CABEÇA	Sim	Sim	Secção 4.3.2
OPÇÕES	Sim	Sim	Secção 4.3.7
POSTAR	Não	Não	Secção 4.3.3
PÔR	Não	Sim	Secção 4.3.4
TRAÇO	Sim	Sim	Secção 4.3.8

8.2. Registro de código de status

O "Registro de Código de Status do Protocolo de Transferência de Hipertexto (HTTP)" define o namespace para o token de código de status de resposta (Seção 6). O registro do código de status é mantido em <http://www.iana.org/assignments/http-status-codes>.

Esta seção substitui o procedimento de registro para códigos de status HTTP anteriormente definidos na Seção 7.1 de [RFC2817].

8.2.1. Procedimento

Um registro DEVE incluir os seguintes campos: o

Código de status (3 dígitos)

- Breve descrição

- Ponteiro para o texto da especificação

Os valores a serem adicionados ao namespace do código de status HTTP requerem a revisão da IETF (consulte [RFC5226], Seção 4.1).

8.2.2. Considerações sobre novos códigos de status

Quando é necessário expressar semântica para uma resposta que não é definida pelos códigos de status atuais, um novo código de status pode ser registrado. Os códigos de status são genéricos; eles são potencialmente aplicáveis a qualquer recurso, não apenas a um tipo de mídia específico, tipo de recurso ou aplicação de HTTP.As tal, é preferível que novos códigos de status sejam registrados em um documento que não seja específico para um único aplicativo.

Os novos códigos de status devem se enquadrar em uma das categorias definidas na Seção 6. Para permitir que os analisadores existentes processem a mensagem de resposta, novos códigos de status não podem proibir uma carga, embora possam exigir um corpo de carga útil de comprimento zero.

As propostas de novos códigos de status que ainda não foram amplamente implantados devem evitar a alocação de um número específico para o código até que haja um consenso claro de que ele será registrado; em vez disso, os primeiros rascunhos podem usar uma notação como "4NN" ou "3N0". "3N9", para indicar a classe do(s) código(s) de status proposto(s) sem consumir um número prematuramente.

A definição de um novo código de status deve explicar as condições de solicitação que causariam uma resposta contendo esse código de status (por exemplo, combinações de campos de cabeçalho de solicitação e/ou método(s)) junto com quaisquer dependências nos campos de cabeçalho de resposta (por exemplo, quais campos são obrigatórios, quais campos podem modificar a semântica e quais semânticas de campo de cabeçalho são refinadas ainda mais quando usadas com o novo código de status).

A definição de um novo código de status deve especificar se é ou não armazenável em cache. Observe que todos os códigos de status podem ser armazenados em cache se a resposta em que ocorrem tiver informações explícitas de atualização; No entanto, os códigos de status definidos como armazenáveis em cache podem ser armazenados em cache sem atualização explícita informação. Da mesma forma, a definição de um código de status pode colocar restrições no comportamento do cache. Consulte [RFC7234] para obter mais informações.

Finalmente, a definição de um novo código de status deve indicar se a carga tem alguma associação implícita com um recurso identificado (Seção 3.1.4.1).

8.2.3. Registros

O registro do código de status foi atualizado com os registros abaixo:

Valor	Descrição	Referência
100	Continuar	Secção 6.2.1
101	Protocolos de comutação	Secção 6.2.2
200	OKEY	Secção 6.3.1
201	Criado	Secção 6.3.2
202	Aceitado	Secção 6.3.3
203	Informações não autorizadas	Secção 6.3.4
204	Sem conteúdo	Secção 6.3.5
205	Redefinir conteúdo	Secção 6.3.6
300	Múltiplas escolhas	Secção 6.4.1
301	Movido permanentemente	Secção 6.4.2
302	Fundar	Secção 6.4.3
303	Ver outros	Secção 6.4.4
305	Usar proxy	Secção 6.4.5
306	(Não utilizado)	Secção 6.4.6
307	Redirecionamento temporário	Secção 6.4.7
400	Solicitação incorreta	Secção 6.5.1
402	Pagamento Obrigatório	Secção 6.5.2
403	Proibido	Secção 6.5.3
404	Não encontrado	Secção 6.5.4
405	Método não permitido	Secção 6.5.5
406	Não aceitável	Secção 6.5.6
408	Tempo limite da solicitação	Secção 6.5.7
409	Conflito	Secção 6.5.8
410	Sumido	Secção 6.5.9
411	Comprimento necessário	Secção 6.5.10
413	Carga útil muito grande	Secção 6.5.11
414	URI muito longo	Secção 6.5.12
415	Tipo de mídia não suportado	Secção 6.5.13
417	Falha na expectativa	Secção 6.5.14
426	Atualização necessária	Secção 6.5.15
500	Erro interno do servidor	Secção 6.6.1
501	Não implementado	Secção 6.6.2
502	Bad Gateway	Secção 6.6.3
503	Serviço indisponível	Secção 6.6.4
504	Tempo limite do gateway	Secção 6.6.5
505	Versão HTTP não suportada	Secção 6.6.6

8.3. Registro de campo de cabeçalho

Os campos de cabeçalho HTTP são registrados no registro "Cabeçalhos de mensagem" localizado em <http://www.iana.org/assignments/message-headers>, conforme definido por [BCP90].

8.3.1. Considerações sobre novos campos de cabeçalho

Os campos de cabeçalho são pares chave:valor que podem ser usados para comunicar dados sobre a mensagem, sua carga, o recurso de destino ou a conexão (ou seja, controle dados). Consulte a Seção 3.2 de [RFC7230] para obter uma definição geral da sintaxe do campo de cabeçalho em mensagens HTTP.

Os requisitos para nomes de campo de cabeçalho são definidos em [BCP90].

Os autores de especificações que definem novos campos são aconselhados a manter o nome o mais curto possível e a não prefixar o nome com "X-", a menos que o campo de cabeçalho nunca seja usado na Internet. (O
O idioma do prefixo "X-" tem sido amplamente mal utilizado na prática; destinava-se a ser usado apenas como um mecanismo para evitar colisões de nomes dentro de software proprietário ou processamento de intranet, uma vez que o prefixo garantiria que nomes privados nunca colidissem com um nome da Internet recém-registrado; consulte [BCP178] para obter mais informações).

Novos valores de campo de cabeçalho normalmente têm sua sintaxe definida usando ABNF ([RFC5234]), usando a extensão definida na Seção 7 de [RFC7230] conforme necessário, e geralmente são restritos ao intervalo de caracteres US-ASCII. Os campos de cabeçalho que precisam de um intervalo maior de caracteres podem usar uma codificação como a definida em [RFC5987].

Os espaços em branco à esquerda e à direita nos valores brutos do campo são removidos na análise do campo (Seção 3.2.4 de [RFC7230]). As definições de campo em que o espaço em branco à esquerda ou à direita nos valores é significativo terão que usar uma sintaxe de contêiner, como a string entre aspas (Seção 3.2.6 de [RFC7230]).

Como as vírgulas (",") são usadas como um delimitador genérico entre valores de campo, eles precisam ser tratados com cuidado se forem permitidos no valor de campo. Normalmente, os componentes que podem conter uma vírgula são protegidos com aspas duplas usando a produção ABNF de cadeia de caracteres entre aspas.

Por exemplo, uma data textual e um URI (qualquer um dos quais pode conter uma vírgula) podem ser transportados com segurança em valores de campo como estes:

```
Exemplo-URI-Field: "http://example.com/a.html,foo",
                   "http://without-a-comma.example.com/"
Example-Date-Field: "Sáb, 04 de maio de 1996", "Qua, 14 de
setembro de 2005"
```

Observe que os delimitadores de aspas duplas quase sempre são usados com a produção de cadeia de caracteres entre aspas; usando

uma sintaxe diferente dentro
aspas duplas provavelmente causarão confusão desnecessária.

Muitos campos de cabeçalho usam um formato que inclui parâmetros nomeados (sem distinção entre maiúsculas e minúsculas) (por exemplo, Content-Type, definido na Seção 3.1.1.5). Permitir a sintaxe sem aspas (token) e entre aspas (cadeia de caracteres entre aspas) para o valor do parâmetro permite que os destinatários usem os componentes do analisador existentes. Ao permitir ambas as formas, o significado de um valor de parâmetro deve ser independente da sintaxe usada para ele (por exemplo, consulte as notas sobre manipulação de parâmetros para tipos de mídia na Seção 3.1.1.1).

Os autores de especificações que definem novos campos de cabeçalho são aconselhados a considerar documentar:

- Se o campo é um único valor ou se pode ser uma lista (delimitado por vírgulas; consulte a Seção 3.2 de [RFC7230]).

  Se ele não usar a sintaxe da lista, documente como tratar as mensagens em que o campo ocorre várias vezes (um padrão sensato seria ignorar o campo, mas isso nem sempre é a escolha certa).

  Observe que intermediários e bibliotecas de software podem combinar várias instâncias de campo de cabeçalho em uma única, apesar da definição do campo não permitir a sintaxe da lista. Um formato robusto permite que os destinatários descubram essas situações (bom exemplo: "Content-Type", pois a vírgula só pode aparecer dentro de strings entre aspas; mau exemplo: "Location", pois uma vírgula pode ocorrer dentro de um URI).

- Em que condições o campo de cabeçalho pode ser usado; por exemplo, apenas em respostas ou solicitações, em todas as mensagens, apenas em respostas a um determinado método de solicitação, etc.

- Se o campo deve ser armazenado por servidores de origem que o entendem em uma solicitação PUT.

- Se a semântica de campo é refinada ainda mais pelo contexto, como por métodos de solicitação ou códigos de status existentes.

- Se é apropriado listar o nome do campo no campo de cabeçalho Conexão (ou seja, se o campo de cabeçalho for salto a salto; consulte a Seção 6.1 de [RFC7230]).

- Sob quais condições os intermediários podem inserir, excluir ou modificar o valor do campo.

- Se é apropriado listar o nome do campo em um campo de cabeçalho de resposta Vary (por exemplo, quando o campo de cabeçalho de solicitação é usado pelo algoritmo de seleção de conteúdo de um servidor de origem; consulte Seção 7.1.4).

- Se o campo de cabeçalho é útil ou admissível em reboques (ver secção 4.1 de [RFC7230]).

- Se o campo de cabeçalho deve ser preservado entre redirecionamentos.

- Se introduz quaisquer considerações de segurança adicionais, como a divulgação de dados relacionados à privacidade.

8.3.2. Registros

O registro "Cabeçalhos de mensagens" foi atualizado com os seguintes registros permanentes:

Nome do campo de cabeçalho	Protocolo	Status	Referência
Aceitar	http	padrão	Secção 5.3.2
Aceitar-Charset	http	padrão	Secção 5.3.3
Aceitar Codificação	http	padrão	Secção 5.3.4
Linguagem de aceitação	http	padrão	Secção 5.3.5
Permitir	http	padrão	Secção 7.4.1
Codificação de conteúdo	http	padrão	Secção 3.1.2.2
Linguagem do conteúdo	http	padrão	Secção 3.1.3.2
Localização do conteúdo	http	padrão	Secção 3.1.4.2
Tipo de conteúdo	http	padrão	Secção 3.1.1.5
Data	http	padrão	Secção 7.1.1.2
Esperar	http	padrão	Secção 5.1.1
De	http	padrão	Secção 5.5.1
Localização	http	padrão	Secção 7.1.2
Max-Forwards	http	padrão	Secção 5.1.2
Versão MIME	http	padrão	Apêndice A.1
Referenciador	http	padrão	Secção 5.5.2
Tentar novamente depois	http	padrão	Secção 7.1.3
Servidor	http	padrão	Seção 7.4.2
Agente do usuário	http	padrão	Secção 5.5.3
Variar	http	padrão	Seção 7.1.4

O controlador de alterações para os registros acima é: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

8.4. Registro de codificação de conteúdo

O "HTTP Content Coding Registry" define o namespace para nomes de codificação de conteúdo (Seção 4.2 de [RFC7230]). O registro de codificação de conteúdo é mantido em <http://www.iana.org/assignments/http-parameters>.

8.4.1. Procedimento

Os registros de codificação de conteúdo DEVEM incluir os

seguintes campos: o Nome

o Descrição

o Ponteiro para o texto da especificação

Os nomes das codificações de conteúdo NÃO DEVEM se sobrepor aos nomes das codificações de transferência (Seção 4 de [RFC7230]), a menos que a transformação de codificação seja idêntica (como é o caso das codificações de compactação definidas na Seção 4.2 de [RFC7230]).

Os valores a serem adicionados a este namespace requerem revisão IETF (consulte a seção
4.1 de [RFC5226]) e DEVE estar em conformidade com a finalidade da codificação de conteúdo definida nesta seção.

8.4.2. Registros

O "HTTP Content Coding Registry" foi atualizado com os registros abaixo:

Nome	Descrição	Referência
identidade	Reservado (sinônimo de "sem codificação" em Aceitar-Codificação)	Secção 5.3.4

9. Considerações de segurança

Esta seção destina-se a informar desenvolvedores, provedores de informações e usuários sobre preocupações de segurança conhecidas relevantes para a semântica HTTP e seu uso para transferência de informações pela Internet.
Considerações relacionadas à sintaxe, análise e roteamento de mensagens são discutidas na Seção 9 de [RFC7230].

A lista de considerações abaixo não é exaustiva. A maioria das

preocupações de segurança relacionadas à semântica HTTP é sobre a proteção de aplicativos do lado do servidor (código por trás da interface HTTP), protegendo o agente do usuário

processamento de cargas úteis recebidas via HTTP, ou uso seguro da Internet em geral, em vez da segurança do protocolo. Várias organizações mantêm informações atuais e links para pesquisas atuais sobre segurança de aplicativos da Web (por exemplo, [OWASP]).

9.1. Ataques baseados em nomes de arquivos e caminhos

Os servidores de origem frequentemente usam seu sistema de arquivos local para gerenciar o mapeamento do URI de solicitação efetivo para representações de recursos. A maioria dos sistemas de arquivos não é projetada para proteger contra nomes de arquivos ou caminhos maliciosos. Portanto, um servidor de origem precisa evitar acessar nomes que tenham um significado especial para o sistema ao mapear o destino da solicitação para arquivos, pastas ou diretórios.

Por exemplo, UNIX, Microsoft Windows e outros sistemas operacionais usam ".." como um componente de caminho para indicar um nível de diretório acima do atual e usam caminhos ou nomes de arquivo especialmente nomeados para enviar dados para dispositivos do sistema. Convenções de nomenclatura semelhantes podem existir em outros tipos de armazenamento Sistemas. Da mesma forma, os sistemas de armazenamento local têm uma tendência irritante de preferir a facilidade de uso à segurança ao lidar com caracteres inválidos ou inesperados, recomposição de caracteres decompostos e normalização de maiúsculas e minúsculas de nomes que não diferenciam maiúsculas de minúsculas.

Os ataques baseados nesses nomes especiais tendem a se concentrar na negação de serviço (por exemplo, dizer ao servidor para ler de uma porta COM) ou na divulgação de arquivos de configuração e origem que não devem ser atendidos.

9.2. Ataques baseados em injeção de comando, código ou consulta

Os servidores de origem geralmente usam parâmetros no URI como um meio de identificar serviços do sistema, selecionar entradas de banco de dados ou escolher dados fonte. No entanto, os dados recebidos em uma solicitação não podem ser Confiável. Um invasor pode construir qualquer um dos elementos de dados da solicitação (método, destino da solicitação, campos de cabeçalho ou corpo) para conter dados que podem ser mal interpretados como um comando, código ou consulta quando passados por uma chamada de comando, interpretador de linguagem ou interface de banco de dados.

Por exemplo, a injeção de SQL é um ataque comum em que linguagem de consulta adicional é inserida em alguma parte dos campos de destino de solicitação ou cabeçalho (por exemplo, Host, Referer, etc.). Se os dados recebidos forem usados diretamente em uma instrução SELECT, a linguagem de consulta poderá ser interpretada como um comando de banco de dados em vez de um valor de cadeia de caracteres simples.

Esse tipo de vulnerabilidade de implementação é extremamente comum, apesar de ser fácil de prevenir.

Em geral, as implementações de recursos devem evitar o uso de dados de solicitação em contextos que são processados ou interpretados como instruções. Os parâmetros devem ser comparados a cadeias de caracteres fixas e atuados como resultado dessa comparação, em vez de serem passados por uma interface que não está preparada para não confiar dados. Os dados recebidos que não são baseados em parâmetros fixos devem ser cuidadosamente filtrados ou codificados para evitar serem mal interpretados.

Considerações semelhantes se aplicam aos dados de solicitação quando eles são armazenados e processados posteriormente, como em arquivos de log, ferramentas de monitoramento ou quando incluídos em um formato de dados que permite scripts inseridos.

9.3. Divulgação de informações pessoais

Os clientes geralmente têm acesso a grandes quantidades de informações pessoais, incluindo informações fornecidas pelo usuário para interagir com recursos (por exemplo, nome do usuário, localização, endereço de e-mail, senhas, chaves de criptografia etc.) e informações sobre a atividade de navegação do usuário ao longo do tempo (por exemplo, histórico, favoritos, etc.). As implementações precisam evitar a divulgação não intencional de informações pessoais.

9.4. Divulgação de informações confidenciais em URIs

Os URIs devem ser compartilhados, não protegidos, mesmo quando identificam recursos seguros. Os URIs geralmente são mostrados em exibições, adicionados a modelos quando uma página é impressa e armazenados em uma variedade de listas de favoritos desprotegidas. Portanto, não é aconselhável incluir informações em um URI que sejam confidenciais, pessoalmente identificáveis ou que representem um risco de divulgação.

Os autores de serviços devem evitar formulários baseados em GET para o envio de dados confidenciais, pois esses dados serão colocados no alvo de solicitação. Muitos servidores, proxies e agentes de usuário existentes registram ou exibem o destino da solicitação em locais onde ele pode ser visível para terceiros Partes. Esses serviços devem usar o envio de formulários baseado em POST.

Como o campo de cabeçalho Referer informa a um site de destino sobre o contexto que resultou em uma solicitação, ele tem o potencial de revelar informações sobre o histórico de navegação imediata do usuário e qualquer informação pessoal que possa ser encontrada no recurso de referência URI. As limitações no campo de cabeçalho Referer são descritas na Seção 5.5.2 para tratar de algumas de suas considerações de segurança.

9.5. Divulgação de fragmento após redirecionamentos

Embora os identificadores de fragmento usados nas referências de URI não sejam enviados em solicitações, os implementadores devem estar cientes de que eles estarão visíveis para o agente do usuário e quaisquer extensões ou scripts executados como resultado do resposta. Em particular, quando ocorre um redirecionamento e o identificador de fragmento da solicitação original é herdado pela nova referência em Local (Seção 7.1.2), isso pode ter o efeito de divulgar o fragmento de um site para outro site. Se o primeiro site usar informações pessoais em fragmentos, ele deve garantir que os redirecionamentos para outros sites incluam um componente de fragmento (possivelmente vazio) para bloquear essa herança.

9.6. Divulgação de informações sobre produtos

Os campos de cabeçalho User-Agent (Seção 5.5.3), Via (Seção 5.7.1 de [RFC7230]) e Servidor (Seção 7.4.2) geralmente revelam informações sobre os sistemas de software do respectivo remetente. Em teoria, isso pode tornar mais fácil para um invasor explorar falhas de segurança conhecidas; Na prática, os invasores tendem a tentar todas as falhas potenciais, independentemente das versões aparentes do software que estão sendo usadas.

Os proxies que servem como um portal por meio de um firewall de rede devem tomar precauções especiais em relação à transferência de informações de cabeçalho que podem identificar hosts por trás do firewall. O campo de cabeçalho Via permite que os intermediários substituam nomes de máquina confidenciais por pseudônimos.

9.7. Impressão digital do navegador

A impressão digital do navegador é um conjunto de técnicas para identificar um agente de usuário específico ao longo do tempo por meio de seu conjunto exclusivo de Características. Essas características podem incluir informações relacionadas ao comportamento do TCP, recursos de recursos e ambiente de script, embora de particular interesse aqui seja o conjunto de características exclusivas que podem ser comunicadas via HTTP. A impressão digital é considerada uma preocupação de privacidade porque permite o rastreamento do comportamento de um agente do usuário ao longo do tempo sem os controles correspondentes que o usuário pode ter sobre outras formas de coleta de dados (por exemplo, cookies). Muitos agentes de usuário de uso geral (ou seja, navegadores da Web) tomaram medidas para reduzir suas impressões digitais.

Há vários campos de cabeçalho de solicitação que podem revelar informações aos servidores que são suficientemente exclusivas para habilitar a impressão digital. O campo de cabeçalho From é o mais óbvio, embora se espere que From só seja enviado quando a autoidentificação for desejada pelo utilizador. Da mesma forma, os

campos de cabeçalho do cookie são deliberadamente

projetado para permitir a reidentificação, portanto, as preocupações com impressões digitais se aplicam apenas a situações em que os cookies são desativados ou restritos pela configuração do agente do usuário.

O campo de cabeçalho User-Agent pode conter informações suficientes para identificar exclusivamente um dispositivo específico, geralmente quando combinado com outras características, especialmente se o user agent enviar detalhes excessivos sobre o sistema do usuário ou Extensões. No entanto, a fonte de informações exclusivas menos esperadas pelos usuários é a negociação proativa (Seção 5.3), incluindo os campos de cabeçalho Accept, Accept-Charset, Accept-Coding e Accept-Language.

Além da preocupação com a impressão digital, o uso detalhado do campo de cabeçalho Accept-Language pode revelar informações que o usuário pode considerar de natureza privada. Por exemplo, a compreensão de um determinado conjunto de idiomas pode estar fortemente correlacionada à associação a uma determinada etnia grupo. Uma abordagem que limita essa perda de privacidade seria um agente de usuário omitir o envio de
Accept-Language, exceto para sites que foram incluídos na lista de permissões, talvez por meio de interação após a detecção de um campo de cabeçalho Vary que indica que a negociação de idioma pode ser útil.

Em ambientes onde os proxies são usados para aumentar a privacidade, os agentes do usuário devem ser conservadores no envio de cabeçalho de negociação proativo Campos. Os agentes de usuário de uso geral que fornecem um alto grau de configurabilidade do campo de cabeçalho devem informar os usuários sobre a perda de privacidade que pode resultar se muitos detalhes forem fornecidos. Como uma medida de privacidade extrema, os proxies podem filtrar os campos de cabeçalho de negociação proativa em solicitações retransmitidas.

10. Confirmações

Ver secção 10 da [RFC7230].

11. Referências

11.1. Referências Normativas

[RFC2045] Freed, N. e N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, novembro de 1996.

[RFC2046] Freed, N. e N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, novembro de 1996.

[RFC2119] Bradner, S., "Palavras-chave para uso em RFCs para indicar níveis de requisitos", BCP 14, RFC 2119, março de 1997.

[RFC3986] Berners-Lee, T., Fielding, R., e L. Masinter, "Uniform Resource Identifier (URI): Sintaxe Genérica", STD 66, RFC 3986, janeiro de 2005.

[RFC4647] Phillips, A., Ed. e M. Davis, Ed., "Correspondência de tags de idioma", BCP 47, RFC 4647, setembro de 2006.

[RFC5234] Crocker, D., Ed. e P. Overell, "BNF aumentado para especificações de sintaxe: ABNF", STD 68, RFC 5234, janeiro de 2008.

[RFC5646] Phillips, A., Ed. e M. Davis, Ed., "Tags para Identificação de Idiomas", BCP 47, RFC 5646, setembro de 2009.

[RFC6365] Hoffman, P. e J. Klensin, "Terminologia Usada na Internacionalização no IETF", BCP 166, RFC 6365, setembro de 2011.

[RFC7230] Fielding, R., Ed. e J. Reschke, Ed., "Protocolo de Transferência de Hipertexto (HTTP / 1.1): Sintaxe e Roteamento de Mensagens", RFC 7230, junho de 2014.

[RFC7232] Fielding, R., Ed. e J. Reschke, Ed., "Protocolo de Transferência de Hipertexto (HTTP / 1.1): Solicitações Condicionais", RFC 7232, Junho de 2014.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., e J. Reschke, Ed., "Protocolo de Transferência de Hipertexto (HTTP/1.1): Solicitações de Intervalo", RFC 7233, junho de 2014.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., e J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, junho de 2014.

[RFC7235] Fielding, R., Ed. e J. Reschke, Ed., "Protocolo de Transferência de Hipertexto (HTTP / 1.1): Autenticação", RFC 7235, junho de 2014.

11.2. Referências informativas

[BCP13]Freed, N., Klensin, J. e T. Hansen, "Especificações de tipo de mídia e procedimentos de registro", BCP 13, RFC 6838, janeiro de 2013.

[BCP178]Saint-Andre, P., Crocker, D. e M. Nottingham, "Descontinuando o prefixo "X-" e construções semelhantes em protocolos de aplicativos", BCP 178, RFC 6648, junho de 2012.

[BCP90]Klyne, G., Nottingham, M. e J. Mogul, "Procedimentos de Registro para Campos de Cabeçalho de Mensagem", BCP 90, RFC 3864, setembro de 2004.

[OWASP]van der Stock, A., Ed., "Um Guia para a Construção de Aplicações Web Seguras e Serviços Web", The Open Web Application Security Project (OWASP) 2.0.1, Julho de 2005, <https://www.owasp.org/>.

[DESCANSO]Fielding, R., "Estilos Arquitetônicos e o Design de Arquiteturas de Software Baseadas em Rede", Tese de doutorado, Universidade da Califórnia, Irvine, setembro de 2000, <http://roy.gbiv.com/pubs/dissertation/top.htm>.

[RFC1945] Berners-Lee, T., Fielding, R., e H. Nielsen, "Protocolo de Transferência de Hipertexto -- HTTP/1.0", RFC 1945, Maio de 1996.

[RFC2049] Freed, N. e N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, novembro de 1996.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H. e T. Berners-Lee, "Protocolo de Transferência de Hipertexto - - HTTP/1.1", RFC 2068, janeiro de 1997.

[RFC2295] Holtman, K. e A. Mutz, "Negociação de Conteúdo Transparente em HTTP", RFC 2295, março de 1998.

[RFC2388] Masinter, L., "Retornando valores de Formulários: multipart / dados de formulário", RFC 2388, agosto de 1998.

[RFC2557] Palme, F., Hopmann, A., Shelness, N., e E. Stefferud, "Encapsulamento MIME de Documentos Agregados, como HTML (MHTML)", RFC 2557, março de 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. e T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, junho de 1999.

[RFC2774] Frystyk, H., Leach, P. e S. Lawrence, "An HTTP Extension Framework", RFC 2774, fevereiro de 2000.

[RFC2817] Khare, R. e S. Lawrence, "Atualizando para TLS dentro do HTTP/1.1", RFC 2817, maio de 2000.

[RFC2978] Freed, N. e J. Postel, "Procedimentos de Registro de

Charset da IANA", BCP 19, RFC 2978, outubro de 2000.

[RFC5226] Narten, T. e H. Alvestrand, "Diretrizes para escrever uma seção de considerações da IANA em RFCs", BCP 26, RFC 5226, maio de 2008.

[RFC5246] Dierks, T. e E. Rescorla, "O protocolo TLS (Transport Layer Security) versão 1.2", RFC 5246, agosto de 2008.

[RFC5322] Resnick, P., "Formato de mensagem da Internet", RFC 5322, outubro de 2008.

[RFC5789] Dusseault, L. e J. Snell, "Método PATCH para HTTP", RFC 5789, março de 2010.

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J. e W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, junho de 2010.

[RFC5987] Reschke, J., "Conjunto de caracteres e codificação de idioma para parâmetros de campo de cabeçalho do protocolo de transferência de hipertexto (HTTP)", RFC 5987, agosto de 2010.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, outubro de 2010.

[RFC6265] Barth, A., "Mecanismo de gerenciamento de estado HTTP", RFC 6265, Abril de 2011.

[RFC6266] Reschke, J., "Uso do campo de cabeçalho de disposição de conteúdo no protocolo de transferência de hipertexto (HTTP)", RFC 6266, junho de 2011.

[RFC7238] Reschke, J., "O Código de Status do Protocolo de Transferência de Hipertexto (HTTP) 308 (Redirecionamento Permanente)", RFC 7238, junho de 2014.

Apêndice A. Diferenças entre HTTP e MIME

O HTTP/1.1 usa muitas das construções definidas para o Formato de Mensagem da Internet [RFC5322] e as Extensões de Correio da Internet Multiuso (MIME) [RFC2045] para permitir que um corpo de mensagem seja transmitido em uma variedade aberta de representações e com campos de cabeçalho extensíveis.
No entanto, o RFC 2045 está focado apenas no e-mail; os aplicativos de HTTP têm muitas características que diferem do e-mail; portanto, o HTTP tem recursos que diferem de MÍMICA. Essas diferenças foram cuidadosamente escolhidas para otimizar o desempenho em conexões binárias, para permitir maior liberdade no uso de novos tipos de mídia, para facilitar as comparações de datas e para reconhecer a prática de alguns dos primeiros servidores e clientes HTTP.

Este apêndice descreve áreas específicas em que o HTTP difere do MIME. Proxies e gateways de e para ambientes MIME estritos precisam estar cientes dessas diferenças e fornecer as conversões apropriadas quando necessário.

A.1. Versão MIME

HTTP não é um protocolo compatível com MIME. No entanto, as mensagens podem incluir um único campo de cabeçalho MIME-Version para indicar qual versão do protocolo MIME foi usada para construir o Mensagem. Uso do
O campo de cabeçalho MIME-Version indica que a mensagem está em total conformidade com o protocolo MIME (conforme definido em [RFC2045]). Os remetentes são responsáveis por garantir a conformidade total (sempre que possível) ao exportar mensagens HTTP para ambientes MIME estritos.

A.2. Conversão para a forma canônica

O MIME requer que uma parte do corpo do correio da Internet seja convertida para a forma canônica antes de ser transferida, conforme descrito na Seção 4 de [RFC2049]. A seção 3.1.1.3 deste documento descreve as formas permitidas para subtipos do tipo de mídia "texto" quando transmitidos por HTTP. [RFC2046] exige que o conteúdo com um tipo de "texto" represente quebras de linha como CRLF e proíbe o uso de CR ou LF fora da quebra de linha Seqüências. O HTTP permite que CRLF, CR nu e LF nu indiquem uma quebra de linha no conteúdo do texto.

Um proxy ou gateway de HTTP para um ambiente MIME estrito deve traduzir todas as quebras de linha dentro dos tipos de mídia de texto descritos na Seção 3.1.1.3 deste documento para a forma canônica RFC 2049 do CRLF. Observe, no entanto, que isso pode ser complicado pela presença de uma codificação de conteúdo e pelo fato de que o HTTP permite o uso de alguns conjuntos de caracteres que não usam os octetos 13 e 10 para representar CR e LF,

respectivamente.

A conversão interromperá todas as somas de verificação criptográficas aplicadas ao conteúdo original, a menos que o conteúdo original já esteja na forma canônica. Portanto, a forma canônica é recomendada para qualquer conteúdo que use essas somas de verificação em HTTP.

A.3. Conversão de formatos de data

O HTTP/1.1 usa um conjunto restrito de formatos de data (Seção 7.1.1.1) para simplificar o processo de data comparação. Proxies e gateways de outros protocolos devem garantir que qualquer campo de cabeçalho Date presente em uma mensagem esteja em conformidade com um dos formatos HTTP/1.1 e reescrever a data, se necessário.

A.4. Conversão de codificação de conteúdo

O MIME não inclui nenhum conceito equivalente ao HTTP/1.1 Cabeçalho de codificação de conteúdo campo. Como isso atua como um modificador no tipo de mídia, os proxies e gateways de protocolos compatíveis com HTTP para MIME devem alterar o valor do campo de cabeçalho Content-Type ou decodificar a representação antes de encaminhar a mensagem. (Algumas aplicações experimentais de Content-Type para correio da Internet usaram um parâmetro de tipo de mídia de "; conversions=<content-coding>" para executar uma função equivalente a Content-Encoding. No entanto, esse parâmetro não faz parte dos padrões MIME).

A.5. Conversão de codificação de transferência de conteúdo

O HTTP não usa o campo Content-Transfer-Encoding do MIME. Proxies e gateways de protocolos compatíveis com MIME para HTTP precisam remover qualquer Content-Transfer-Encoding antes de entregar a mensagem de resposta a um cliente HTTP.

Proxies e gateways de protocolos compatíveis com HTTP para MIME são responsáveis por garantir que a mensagem esteja no formato correto e codificar para transporte seguro nesse protocolo, onde o "transporte seguro" é definido pelas limitações do protocolo que está sendo usado. Esse proxy ou gateway deve transformar e rotular os dados com uma codificação de transferência de conteúdo apropriada se isso melhorar a probabilidade de transporte seguro pelo protocolo de destino.

A.6. Limitações de MHTML e comprimento de linha

As implementações HTTP que compartilham código com implementações MHTML [RFC2557] precisam estar cientes das limitações de comprimento de linha MIME. Como o HTTP não tem essa limitação, o HTTP não dobra linhas longas. As mensagens MHTML transportadas por HTTP seguem todas as convenções do MHTML, incluindo limitações de comprimento de linha e dobramento, canonização, etc., uma vez que o HTTP transfere os corpos das mensagens como

payload e, além do tipo "multipart/byteranges" (Apêndice A de [RFC7233]), não interpreta o conteúdo ou quaisquer linhas de cabeçalho MIME que possam estar contidas nele.

Apêndice B. Alterações do RFC 2616

As principais alterações nesta revisão foram de natureza editorial: extrair a sintaxe do sistema de mensagens e particionar a semântica HTTP em documentos separados para os recursos principais, solicitações condicionais, solicitações parciais, armazenamento em cache e autenticação. A linguagem de conformidade foi revisada para direcionar claramente os requisitos e a terminologia foi aprimorada para distinguir a carga útil das representações e as representações dos recursos.

Um novo requisito foi adicionado para que a semântica incorporada em um URI seja desabilitada quando essa semântica for inconsistente com o método de solicitação, pois essa é uma causa comum de falha de interoperabilidade. (Seção 2)

Um algoritmo foi adicionado para determinar se uma carga está associada a um identificador. (Seção 3.1.4.1)

O conjunto de caracteres padrão de ISO-8859-1 para tipos de mídia de texto foi removido; O padrão agora é o que quer que a definição do tipo de mídia diga. Da mesma forma, o tratamento especial de ISO-8859-1 foi removido do cabeçalho Accept-Charset campo. (Seção 3.1.1.3 e Seção 5.3.3)

A definição de Content-Location foi alterada para não afetar mais o URI base para resolver referências de URI relativas, devido ao suporte de implementação inadequado e ao efeito indesejável de potencialmente quebrar links relativos em recursos negociados por conteúdo.
(Seção 3.1.4.2)

Para ser consistente com o algoritmo de análise neutro de método de [RFC7230], a definição de GET foi relaxada para que as solicitações possam ter um corpo, mesmo que um corpo não tenha significado para GET. (Seção 4.3.1)

Os servidores não são mais obrigados a lidar com todos os campos de cabeçalho Content-* e o uso de Content-Range foi explicitamente proibido em solicitações PUT. (Seção 4.3.4)

A definição do método CONNECT foi movida de [RFC2817] para esta especificação. (Seção 4.3.6)

Os métodos de solicitação OPTIONS e TRACE foram definidos como seguros. (Seção 4.3.7 e Seção 4.3.8)

O mecanismo de extensão do campo de cabeçalho Expect foi removido devido a implementações quebradas amplamente implantadas. (Seção 5.1.1)

O campo de cabeçalho Max-Forwards foi restrito aos métodos OPTIONS e TRACE; Anteriormente, os métodos de extensão também poderiam tê-lo usado. (Seção 5.1.2)

O URI "about:blank" foi sugerido como um valor para o campo de cabeçalho Referer quando nenhum URI de referência é aplicável, o que distingue esse caso de outros em que o campo Referer não é enviado ou foi Removido. (Seção 5.5.2)

Os seguintes códigos de status agora podem ser armazenados em cache (ou seja, podem ser armazenados e reutilizados por um cache sem informações explícitas de atualização presentes): 204, 404, 405, 414, 501. (Seção 6)

A descrição do status 201 (Criado) foi alterada para permitir a possibilidade de que mais de um recurso tenha sido criado. (Seção 6.3.2)

A definição de 203 (Informações Não Autorizadas) foi ampliada para incluir também casos de transformações de carga útil. (Seção 6.3.4)

O conjunto de métodos de solicitação que são seguros para redirecionar automaticamente não está mais fechado; Os agentes do usuário são capazes de fazer essa determinação com base no método de solicitação semântica. Os códigos de status de redirecionamento 301, 302 e 307 não têm mais requisitos normativos sobre cargas de resposta e usuário interação. (Seção 6.4)

Os códigos de status 301 e 302 foram alterados para permitir que os agentes do usuário reescrevam o método de POST para GET. (Secções 6.4.2 e 6.4.3)

A descrição do código de status 303 (Consulte Outros) foi alterada para permitir que ele seja armazenado em cache se informações de atualização explícitas forem fornecidas, e uma definição específica foi adicionada para uma resposta 303 ao GET. (Seção 6.4.4)

O código de status 305 (Usar proxy) foi preterido devido a preocupações de segurança relacionadas à configuração em banda de um proxy. (Seção 6.4.5)

O código de status 400 (Solicitação Incorreta) foi relaxado para que não se limite a erros de sintaxe. (Seção 6.5.1)

O código de status 426 (atualização necessária) foi incorporado a partir de [RFC2817]. (Seção 6.5.15)

O destino dos requisitos em HTTP-date e o campo de cabeçalho Date foram reduzidos aos sistemas que geram a data, em vez de todos os sistemas enviarem uma data. (Seção 7.1.1)

A sintaxe do campo de cabeçalho Location foi alterada para permitir todas as referências de URI, incluindo referências relativas e fragmentos, juntamente com alguns esclarecimentos sobre quando o uso de fragmentos não seria apropriado. (Seção 7.1.2)

Allow foi reclassificado como um campo de cabeçalho de resposta, removendo a opção de especificá-lo em uma solicitação PUT. Os requisitos relacionados ao conteúdo do Allow foram relaxados; Da mesma forma, os clientes não são obrigados a confiar sempre em seu valor. (Seção 7.4.1)

Um Registro de Método foi definido. (Seção 8.1)

O Registro de Código de Status foi redefinido por esta especificação; anteriormente, era definido na Seção 7.1 de [RFC2817]. (Seção 8.2)

O registro de codificações de conteúdo foi alterado para exigir a revisão do IETF. (Seção 8.4)

O campo de cabeçalho Content-Disposition foi removido, pois agora é definido por [RFC6266].

O campo de cabeçalho Content-MD5 foi removido porque foi implementado de forma inconsistente em relação a respostas parciais.

Apêndice C. ABNF importado

As seguintes regras principais são incluídas por referência, conforme definido no Apêndice B.1 de [RFC5234]: ALPHA (letras), CR (retorno de carro), CRLF (CR LF), CTL (controles), DIGIT (decimal 0-9), DQUOTE (aspas duplas), HEXDIG (hexadecimal 0-9/AF/af), HTAB (guia horizontal), LF (alimentação de linha), OCTET (qualquer sequência de dados de 8 bits), SP (espaço) e VCHAR (qualquer caractere US-ASCII visível).

As regras abaixo são definidas em [RFC7230]:

```
  BWS= <BWS, ver [RFC7230], Secção 3.2.3>
  OWS= <OWS, ver [RFC7230], Secção 3.2.3>
  RWS= <RWS, ver [RFC7230], Secção 3.2.3>
  Referência de URI = <Referência de URI, consulte [RFC7230],
  Seção 2.7> absolute-URI= <absolute-URI, consulte [RFC7230],
  Seção 2.7>    comment= <comentário, ver [RFC7230], Secção
  3.2.6>        field-name= <comentário, consulte [RFC7230],
  Seção 3.2>    URI parcial = <URI parcial, consulte
  [RFC7230], Seção 2.7>
```

string-asped-= <string-quoted, consulte [RFC7230], Seção 3.2.6> token= <token, consulte [RFC7230], Seção 3.2.6>

Apêndice D. ABNF coletado

No ABNF coletado abaixo, as regras da lista são expandidas de acordo com a Seção 1.2 de [RFC7230].

```
Aceitar = [ ( "," / ( intervalo de mídia [ accept-params ] ) ) *(
 OWS "," [ OWS ( media-range [ accept-params ] ) ] ) ] ]
Accept-Charset = *( "," OWS ) ( ( charset / "*" ) [ peso ] ) *( OWS
 "," [ OWS ( ( charset / "*" ) [ peso ] ) ] ) )
Aceitar-Codificação = [ ( "," / ( codificações [ peso ] ) ) *( OWS
 "," [ OWS ( codificações [ peso ] ] ] ) ] ] ]
Accept-Language = *( "," OWS ) ( language-range [ weight ] ) *( OWS
 "," [ OWS ( language-range [ weight ] ) ] ) )
Permitir = [ ( "," / método ) *( OWS "," [ método OWS ]

) ] BWS = <BWS, ver [RFC7230], Seção 3.2.3>

Codificação de conteúdo = *( "," OWS ) codificação de conteúdo
 *( OWS "," [ OWS codificação de conteúdo ] )
Conteúdo-Idioma = *( "," OWS ) marca-idioma *( OWS "," [ OWS
 marca-idioma ] )
Local do conteúdo = URI absoluto / URI
parcial Tipo de conteúdo = tipo de mídia

Data = data HTTP

Expect = "100-continue"

From = caixa de correio

GMT = %x47.4D.54 ; GMT

Data-HTTP = data-fix-IMF / data-obs-

IMF-fixdate = nome-do-dia "," SP data1 SP hora-do-dia SP GMT

Localização = URI-reference

Max-Forwards = 1*DÍGITO

OWS = <OWS, ver [RFC7230], Secção 3.2.3>

RWS = <RWS, ver [RFC7230], Secção 3.2.3>
Referer = URI absoluto / URI parcial
Repetir depois = data HTTP / segundos
de atraso
```

```
   Servidor = produto *( RWS ( produto / comentário ) ) )

   URI-reference = <URI-reference, consulte [RFC7230], Seção
   2.7> User-Agent = produto *( RWS ( produto / comentário ) )
   )

   Variar = "*" / ( *( "," OWS ) nome do campo *( OWS "," [ OWS nome do
   campo ]
    ) )

   URI absoluto = <URI absoluto, consulte [RFC7230], Seção 2.7>
   aceitar-ext = OWS ";" Token OWS [ "=" ( token / string-entre
   aspas ) ] accept-params = weight *accept-ext
   asctime-date = dia-nome SP data3 SP hora do dia SP ano

   conjunto de caracteres = token
   codings = content-coding / "identity" / "*"
   comment = <comment, ver [RFC7230], Seção 3.2.6>
   content-coding = token

   data1 = dia SP mês SP ano data2
   = dia "-" mês "-" 2DÍGITO
   data3 = mês SP ( 2DIGIT / ( SP DIGIT ) )
   dia = 2DIGIT
   Nome do dia = %x4d.6f.6a ; Mente
    / %x54.75.65 ; Ter
    / %x57.65.64 ; Casar
    / %x54.68.75 ; Qui
    / %x46.72.69 ; Sex
    / %x53.61.74 ; Sáb
    / %x53.75.6E ; Sol
   dia-nome-l = %x4D.6F.6E.64.61.79 ; Segunda-feira
    / %x54.75.65.73.64.61.79 ; Terça-feira
    / %x57.65.64.6E.65.73.64.61.79 ; Quarta-feira
    / %x54.68.75.72.73.64.61.79 ; Quinta-feira
    / %x46.72.69.64.61.79 ; Sexta-feira
    / % x53.61.74.75.72.64.61.79 ; Sábado
    / %x53.75.6E.64.61.79 ; Domingo
   segundos de atraso = 1*DÍGITO

   nome-do-campo = <comentário, consulte [RFC7230],

   Seção 3.2> hora = 2DIGIT

   intervalo de idiomas = <intervalo de idiomas, consulte
   [RFC4647], Seção 2.1> marca de idioma = <Marca de idioma,
   consulte [RFC5646], Seção 2.1>

   mailbox = <mailbox, consulte [RFC5322], Seção 3.4>
   intervalo de mídia = ( "*/*" / ( tipo "/*" ) / ( tipo "/" subtipo ) )
    *( OWS ";" Parâmetro OWS )
```

```
   tipo de mídia = tipo "/" subtipo *( OWS ";" Parâmetro
   OWS ) método = token
   minuto = 2 DÍGITOS
   mês = %x4A.61.6E ; Jan
    / %x46.65.62 ; Fevereiro
    / %x4D.61.72 ; Março
    / %x41.70.72 ; Abr
    / %x4D.61.79 ; Maio
    / %x4A.75.6E ; Jun
    / %x4A.75.6C ; Jul
    / %x41.75.67 ; Agosto
    / %x53.65.70 ; Setembro
    / %x4F.63.74 ; Outubro
    / %x4E.6F.76 ; Novembro
    / %x44.65.63 ; Dezembro

   obs-date=rfc850-date/asctime-date

   parâmetro = token "=" ( token / string entre aspas )
   URI parcial = <URI parcial, consulte [RFC7230], Seção 2.7>
   produto = token [ "/" versão do produto ]
   versão do produto = token
   string-aspas = <string-entre aspas, consulte [RFC7230], Seção
   3.2.6> qvalue = ( "0" [ "." *3DIGIT ] ) / ( "1" [ "." *3"0" ] )

   rfc850-data = dia-nome-l "," SP data2 SP hora do dia SP GMT

   segundo = 2DIGIT
   subtipo = token

   hora do dia = hora ":" minuto ":" segundo
   token = <token, consulte [RFC7230], Seção
   3.2.6> tipo = token

   peso = OWS ";" OWS "q=" qvalue

   ano = 4DIGIT
```

Índice

Do campo de cabeçalho 44

Endereços dos Autores

Roy T. Fielding (editor)
Adobe Systems Incorporated
Avenida do Parque 345
São José CA95110
EUA

Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster NW48155
Alemanha

E-mail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/