Grupo de trabalho de rede
Solicitação de comentários: 3875
Foundation: Informativa

D.Robinson
Categoria da Apache Software
K.
Coar
Fundação Apache Software
Outubro de 2004

A Interface de Gateway Comum (CGI) Versão 1.1

Status deste memorando

Este memorando fornece informações para a comunidade da Internet. Ele não especifica nenhum tipo de padrão de Internet. A distribuição deste memorando é ilimitada.

Aviso de direitos autorais

Nota IESG

Este documento não é candidato a nenhum nível de Padrão da Internet. A IETF se isenta de qualquer conhecimento da adequação deste documento para qualquer finalidade e, em particular, observa que não houve revisão da IETF para questões como segurança, controle de congestionamento ou interação inadequada com protocolos implantados. O Editor RFC optou por publicar este documento a seu critério. Os leitores deste documento devem ter cautela ao avaliar o seu valor para implementação e implantação.

Resumo

A Common Gateway Interface (CGI) é uma interface simples para executar programas externos, software ou gateways em um servidor de informações de maneira independente de plataforma. Atualmente, os servidores de informações suportados são servidores HTTP.

A interface está em uso pela World Wide Web (WWW) desde 1993. Esta especificação define os parâmetros de "prática atual" da interface "CGI/1.1" desenvolvida e documentada no Centro Nacional de Aplicações de Supercomputação dos EUA. Este documento também define o uso da interface CGI/1.1 no UNIX(R) e outros sistemas similares.

Índice

1 Introdução

1.1 Propósito

A Common Gateway Interface (CGI) [22] permite que um servidor HTTP [1], [4] e um script CGI compartilhem a responsabilidade de responder às solicitações do cliente. A solicitação do cliente compreende um Uniform Resource Identifier (URI) [11], um método de solicitação e diversas informações auxiliares sobre a solicitação fornecidas pelo protocolo de transporte.

O CGI define os parâmetros abstratos, conhecidos como metavariáveis, que descrevem a solicitação de um cliente. Juntamente com uma interface de programador concreta, isso especifica uma interface independente de plataforma entre o script e o servidor HTTP.

O servidor é responsável por gerenciar problemas de conexão, transferência de dados, transporte e rede relacionados à solicitação do cliente, enquanto o script CGI trata dos problemas da aplicação, como acesso a dados e processamento de documentos.

1.2 Requisitos

As palavras-chave ‘DEVE’, ‘NÃO DEVE’, ‘REQUERIDO’, ‘DEVE’, ‘NÃO DEVE’, ‘DEVE’, ‘NÃO DEVE’, ‘RECOMENDADO’, ‘MAIO’ e ‘OPCIONAL’ neste documento devem ser interpretados conforme descrito em BCP 14, RFC 2119 [3].

Uma implementação não é compatível se não satisfaz um ou mais dos requisitos “obrigatórios” dos protocolos que implementa. Uma implementação que satisfaz todos os requisitos “obrigatórios” e “deveria” para as suas características é considerada “incondicionalmente compatível”; aquele que satisfaz todos os requisitos “obrigatórios”, mas nem todos os requisitos “deveria” para suas características, é considerado “conforme condicionalmente”.

1.3 Especificações

Nem todas as funções e características do CGI estão definidas na parte principal desta especificação. As seguintes frases são usadas para descrever os recursos que não são especificados:

‘system-deserabaixo'
: O recurso pode diferir entre sistemas, mas deve ser o mesmo para diferentes implementações que utilizam o mesmo sistema. Um sistema geralmente identificará uma classe de sistemas operacionais. Alguns sistemas são definidos na seção 7 deste documento. Novos sistemas poderão ser definidos por novas especificações sem revisão deste documento.

'implementação deserabaixo'
: O comportamento do recurso pode variar de implementação para implementação; uma implementação específica deve documentar seu comportamento.

1.4 Terminologia

Esta especificação usa muitos termos definidos na especificação HTTP/1.1 [4]; no entanto, os seguintes termos são aqui utilizados num sentido que pode não estar de acordo com as suas definições nesse documento, ou com o seu significado comum.

'metavariável'
: Um parâmetro nomeado que transporta informações do servidor para o script. Não é necessariamente uma variável no ambiente do sistema operacional, embora seja a implementação mais comum.

'roteiro'
: O software que é invocado pelo servidor de acordo com esta interface. Não precisa ser um programa independente, mas pode ser uma biblioteca compartilhada ou carregada dinamicamente, ou mesmo uma sub-rotina no servidor. Pode ser um conjunto de declarações interpretadas em tempo de execução, como o termo 'script' é frequentemente entendido, mas isso não é um requisito e dentro do contexto desta especificação o termo tem a definição mais ampla declarada.

'servidor'
: O programa aplicativo que invoca o script para atender às solicitações do cliente.

2 Convenções Notacionais e Gramática Genérica

2.1 BNF aumentado

Todos os mecanismos especificados neste documento são descritos em prosa e em uma forma Backus-Naur aumentada (BNF) semelhante à usada pela RFC 822 [13]. Salvo indicação em contrário, os elementos diferenciam maiúsculas de minúsculas. Este BNF aumentado contém as seguintes construções:

nome = definição
: O nome de uma regra e sua definição são separados pelo caractere de igual ('='). O espaço em branco só é significativo porque as linhas de continuação de uma definição são recuadas.

"literal"
: Aspas duplas (") circundam o texto literal, exceto aspas literais, que são colocadas entre colchetes angulares ('<' e '>').

regra1 | regra2
: As regras alternativas são separadas por uma barra vertical ('|').

(regra1 regra2 regra3)
: Os elementos entre parênteses são tratados como um único elemento.

*regra
: Uma regra precedida por um asterisco ('*') pode ter zero ou mais ocorrências. O formulário completo é '*n**eu* regra' indicando pelo menos *n* e no máximo *eu* ocorrências da regra. *n* e *eu* são valores decimais

opcionais com valores padrão de 0 e infinito, respectivamente.

[regra] Um elemento entre colchetes ('[' e ']') é opcional e equivale a '*1 regra'.

Regra N
Uma regra precedida por um número decimal representa exatamente N ocorrências da regra. É equivalente à 'regra N*N'.

2.2 Regras Básicas

Esta especificação usa uma gramática semelhante à BNF definida em termos de caracteres. Ao contrário de muitas especificações que definem os bytes permitidos por um protocolo, aqui cada literal da gramática corresponde ao caractere que representa. A forma como esses caracteres são representados em termos de bits e bytes dentro de um sistema é definida pelo sistema ou especificada no contexto específico. A única exceção é a regra 'OCTET', definida abaixo.

As regras a seguir são usadas em toda esta especificação para descrever construções básicas de análise.

```
    alfa           = alfa baixo | hialfa
    alfa baixo     = "um" | "b" | "c" | "d" | "e" | "f" | "g" | "h" |
                     "eu" | "j" | "k" | "eu" | "eu" | "n" | "o" | "p" |
                      "q" | "r" | "é" | "t" | "em" | "em" | "Em" | "x" |
                     "e" | "Com
                           "
    hialfa         = "UM" | "B" | "C" | "D" | "E" | "F" | "G" | "H" |
                     "EU" | "J" | "K" | "EU" | "M" | "N" | "O" | "P" |
                      "Q" | "R" | "S" | "T" | "EM" | "V" | "EM" | "X" |
                     "E" | "COM
                           "
    dígito         =  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
                      "8" | "9"
    alfanum        = alfa | dígito
    OCTETO         = <qualquer byte de 8 bits>
    CARACTERIZAÇÃO = alfa | dígito | separador | "!" | "#" | "$" |
                      "%" | "&" | "'" | "*" | "+" | "-" | "." | "`" |
                      "^" | "_" | "{" | "|" | "}" | "~" | CTL
    CTL            = <qualquer caractere de controle>
    SP             = <caractere de espaço>
    HT             = <caractere de tabulação horizontal>
    Holanda        = <nova linha>
    LWSP           = SP | HT | Holanda
    separador      = "(" | ")" | "<" | ">" | "@" | "," | ";" | ":" |

ficha          =  1*<qualquer CHAR exceto CTLs ou separadores>
string entre   =  <"> *qdtext <">
aspas
qdtexto        =  <qualquer CHAR exceto <"> e CTLs, mas incluindo
                  LWSP>
```

```
TEXTO          =  <qualquer caractere imprimível>
                      "\" | <"> | "/" | "[" | "]" | "?" | "=" | "{" |
                      "}" | SP | HT
```

Observe que a nova linha (NL) não precisa ser um único caractere de controle, mas pode ser uma sequência de caracteres de controle. Um sistema PODE definir TEXT como um conjunto maior de caracteres do que <qualquer CHAR excluindo CTLs, mas incluindo LWSP>.

2.3 Codificação de URL

Algumas variáveis e construções usadas aqui são descritas como sendo 'codificadas por URL'. Esta codificação é descrita na seção 2 da RFC 2396 [2]. Em uma string codificada em URL, uma sequência de escape consiste em um caractere de porcentagem ("%") seguido por dois dígitos hexadecimais, onde os dois dígitos hexadecimais formam um octeto. Uma sequência de escape representa o caracter gráfico que tem o octeto como código dentro do conjunto de caracteres codificados US-ASCII [9], se existir. Atualmente não há nenhuma provisão na sintaxe do URI para identificar qual conjunto de caracteres os códigos não-ASCII representam, portanto o CGI lida com esse problema de forma ad-hoc.

Observe que alguns caracteres inseguros (reservados) podem ter semântica diferente quando codificados. A definição de quais caracteres não são seguros depende do contexto; consulte a seção 2 da RFC 2396 [2], atualizada pela RFC 2732 [7], para obter um tratamento oficial. Esses caracteres reservados são geralmente usados para fornecer estrutura sintática à cadeia de caracteres, por exemplo, como separadores de campos. Em todos os casos, a string é processada primeiro em relação a quaisquer caracteres reservados presentes e, em seguida, os dados resultantes podem ser decodificados por URL, substituindo as sequências de escape "%" por seus valores de caracteres.

Para codificar uma sequência de caracteres, todos os caracteres reservados e proibidos são substituídos pelas sequências de escape "%" correspondentes. A string pode então ser usada na montagem de um URI. Os caracteres reservados variam de contexto para contexto, mas sempre serão extraídos deste conjunto:

```
reservado = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
            | "[" | "]"
```

Os dois últimos caracteres foram adicionados pela RFC 2732 [7]. Em qualquer contexto específico, um subconjunto destes caracteres será reservado; os outros caracteres deste conjunto NÃO DEVEM ser codificados quando uma string é codificada por URL nesse contexto. Outras regras básicas usadas para descrever a sintaxe do URI são:

```
hexadecimal =  dígito | "A" | "B" | "C" | "D" | "E" | "F" | "um" |
               "b" | "c" | "d" | "e" | "f"
escapou     =  "%" hexadecimal hexadecimal
sem         =  alfa | dígito | marca
reservas
marca       =  "-" | "_" | "." | "!" | "~" | "*" | "'" | "(" | ")"
```

3 Invocando o script

3.1 Responsabilidades do servidor

O servidor atua como um gateway de aplicativo. Ele recebe a solicitação do cliente, seleciona um script CGI para tratar a solicitação, converte a solicitação do cliente em uma solicitação CGI, executa o script e converte a resposta CGI em uma resposta para o cliente. Ao processar a solicitação do cliente, ele é responsável pela implementação de qualquer protocolo

ou autenticação e segurança em nível de transporte. O servidor PODE também funcionar de forma "não transparente", modificando a solicitação ou resposta para fornecer algum serviço adicional, como transformação do tipo de mídia ou redução de protocolo.

O servidor DEVE realizar traduções e conversões de protocolo nos dados de solicitação do cliente exigidos por esta especificação. Além disso, o servidor mantém sua responsabilidade perante o cliente de estar em conformidade com o protocolo de rede relevante, mesmo se o script CGI não estiver em conformidade com esta especificação.

Se o servidor estiver aplicando autenticação à solicitação, NÃO DEVE executar o script, a menos que a solicitação passe por todos os controles de acesso definidos.

3.2 Seleção de roteiro

O servidor determina qual CGI é o script a ser executado com base em um URI de formato genérico fornecido pelo cliente. Este URI inclui um caminho hierárquico com componentes separados por "/". Para qualquer solicitação específica, o servidor identificará todo ou parte inicial desse caminho com um script individual, colocando assim o script em um ponto específico na hierarquia de caminhos. O restante do caminho, se houver, é um identificador de recurso ou sub-recurso a ser interpretado pelo script.

Informações sobre esta divisão do caminho estão disponíveis para o script nas metavariáveis, descritas abaixo. O suporte para esquemas URI não hierárquicos está fora do escopo desta especificação.

3.3 O script-URI

O mapeamento do URI de solicitação do cliente para a escolha do script é definido pela implementação específica do servidor e sua configuração. O servidor pode permitir que o script seja identificado com um conjunto de diversas hierarquias de caminhos de URI diferentes e, portanto, tem permissão para substituir o URI por outros membros deste conjunto durante o processamento e geração das metavariáveis. O servidor

1. PODE preservar o URI na solicitação específica do cliente; ou
2. PODE selecionar um URI canônico do conjunto de valores possíveis para cada script; ou
3. ele pode implementar qualquer outra seleção de URI do conjunto.

A partir das metavariáveis assim geradas, um URI, o 'Script-URI', pode ser construído. Isto DEVE ter a propriedade de que se o cliente tivesse acessado este URI, então o script teria sido executado com os mesmos valores para as metavariáveis SCRIPT_NAME, PATH_INFO e QUERY_STRING. O Script-URI tem a estrutura de um URI genérico conforme definido na seção 3 da RFC 2396 [2], com a exceção de que parâmetros de objetos e identificadores de fragmentos não são permitidos. Os vários componentes do Script-URI são definidos por algumas das metavariáveis (veja abaixo);

```
script-URI = <esquema> "://" <nome do servidor> ":" <porta do servidor>
             <caminho do script> <caminho extra> "?" <string de
             consulta>
```

onde <scheme> é encontrado em SERVER_PROTOCOL, <server-name>, <server-port> e <query-string> são os valores das respectivas metavariáveis. Os valores SCRIPT_NAME e PATH_INFO, codificados em URL com ";", "=" e "?" reservado, forneça <script-path> e <extra-path>. Consulte a seção 4.1.5 para obter mais informações sobre a metavariável PATH_INFO.

O esquema e o protocolo não são idênticos, pois o esquema identifica o método de acesso além do protocolo de aplicação. Por exemplo, um recurso acessado usando Transport Layer Security (TLS) [14] teria um URI de solicitação com um esquema de `https` ao usar o protocolo HTTP [19]. CGI/1.1 não fornece meios genéricos para o script reconstruir isso e, portanto, o Script-URI conforme definido inclui o protocolo base usado. No entanto, um script PODE fazer uso de metavariáveis específicas do esquema para deduzir melhor o esquema URI.

Observe que esta definição também permite a construção de URIs que invocariam o script com quaisquer valores permitidos para as informações do caminho ou string de consulta, modificando os componentes apropriados.

3.4 Execução

O script é invocado de maneira definida pelo sistema. A menos que especificado de outra forma, o arquivo que contém o script será invocado como um programa executável. O servidor prepara a solicitação CGI conforme descrito na seção 4; isso inclui as metavariáveis de solicitação (imediatamente disponíveis para o script na execução) e os dados da mensagem de solicitação. Os dados da solicitação não precisam estar imediatamente disponíveis para o script; o script pode ser executado antes que todos esses dados sejam recebidos do cliente pelo servidor. A resposta do script é retornada ao servidor conforme descrito nas seções 5 e 6.

No caso de uma condição de erro, o servidor pode interromper ou encerrar a execução do script a qualquer momento e sem aviso prévio. Isso poderia ocorrer, por exemplo, no caso de falha no transporte entre o servidor e o cliente; portanto, o script DEVE estar preparado para lidar com encerramentos anormais.

4 A solicitação CGI

As informações sobre uma solicitação vêm de duas fontes diferentes; as metavariáveis da solicitação e qualquer corpo da mensagem associado.

4.1 Solicitar metavariáveis

As metavariáveis contêm dados sobre a solicitação passada do servidor para o script e são acessadas pelo script de maneira definida pelo sistema. Metavariáveis são identificadas por nomes que não diferenciam maiúsculas de minúsculas; não pode haver duas variáveis diferentes cujos nomes diferem apenas em maiúsculas e minúsculas. Aqui eles são mostrados usando uma representação canônica de letras maiúsculas mais sublinhado ("_"). Um determinado sistema pode definir uma representação diferente.

```
nome-meta-variável = "AUTH_TYPE" | "CONTENT_LENGTH" |
```

```
"CONTENT_TYPE" | "GATEWAY_INTERFACE" |
"PATH_INFO" | "PATH_TRANSLATED" |
```

```
                        "QUERY_STRING" | "REMOTE_ADDR" |
                        "REMOTE_HOST" | "REMOTE_IDENT" |
                        "REMOTE_USER" | "REQUEST_METHOD" |
                        "SCRIPT_NAME" | "SERVER_NAME" |
                        "SERVER_PORT" | "SERVER_PROTOCOL" |
                        "SERVIDOR_SOFTWARE" | esquema |
                        nome-var-protocolo | nome-var-extensão
nome-var-protocolo  = (protocolo | esquema) "_" nome-var
esquema             = alfa *( alfa | dígito | "+" | "-" | "." )
nome-var            = token
extensão-var-nome = token
```

Metavariáveis com o mesmo nome de um esquema e nomes que começam com o nome de um protocolo ou esquema (por exemplo, HTTP_ACCEPT) também são definidas. O número e o significado destas variáveis podem mudar independentemente desta especificação. (Veja também a seção 4.1.18.)

O servidor PODE definir metavariáveis de extensão adicionais definidas pela implementação, cujos nomes DEVEM ser prefixados com “X_”.

Esta especificação não faz distinção entre valores de comprimento zero (NULL) e valores ausentes. Por exemplo, um script não consegue distinguir entre as duas solicitações `http://host/script` e `http://host/script?` como em ambos os casos a metavariável QUERY_STRING seria NULL.

```
meta-variável-valor = "" | 1*<TEXT, CHAR ou tokens de valor>
```

Uma metavariável opcional pode ser omitida (não definida) se seu valor for NULL. Os valores de metavariáveis DEVEM ser considerados sensíveis a maiúsculas e minúsculas, salvo indicação em contrário. A representação dos caracteres nas metavariáveis é definida pelo sistema; o servidor DEVE converter valores para essa representação.

4.1.1 AUTH_TYPE

A variável AUTH_TYPE identifica qualquer mecanismo utilizado pelo servidor para autenticar o usuário. Ele contém um valor que não diferencia maiúsculas de minúsculas definido pelo protocolo do cliente ou pela implementação do servidor.

Para HTTP, se a solicitação do cliente exigir autenticação para acesso externo, o servidor DEVE definir o valor desta variável do token 'auth-scheme' no campo do cabeçalho de autorização da solicitação.

```
AUTH_TYPE       = "" | esquema de autenticação
esquema de autenticação   = "Básico" | "Digerir" |
extensão-auth extensão-auth = token
```

Esquemas de autenticação de acesso HTTP são descritos na RFC 2617 [5].

4.1.2 CONTENT_LENGTH

A variável CONTENT_LENGTH contém o tamanho do corpo da mensagem anexado à solicitação, se houver, em número decimal de octetos. Se nenhum dado estiver anexado, então NULL (ou não definido).

```
CONTENT_LENGTH = "" | 1 * dígito
```

O servidor DEVE definir esta metavariável se e somente se a solicitação for acompanhada por uma entidade do corpo da mensagem. O valor CONTENT_LENGTH deve refletir o comprimento do corpo da mensagem após o servidor ter removido quaisquer codificações de transferência ou de conteúdo.

4.1.3 CONTENT_TYPE

Se a solicitação incluir um corpo de mensagem, a variável CONTENT_TYPE será definida como o Internet Media Type [6] do corpo da mensagem.

```
CONTENT_TYPE  =  "" | tipo de mídia
tipo de mídia =  digite "/" subtipo *( ";" parâmetro)
tipo          =  ficha
subtipo       =  ficha
parâmetro     =  atributo "=" valor
atributo      =  ficha
valor         =  ficha | string entre aspas
```

Os nomes de atributos de tipo, subtipo e parâmetro não diferenciam maiúsculas de minúsculas. Os valores dos parâmetros podem diferenciar maiúsculas de minúsculas. Os tipos de mídia e seu uso em HTTP são descritos na seção 3.7 da especificação HTTP/1.1 [4].

Não há valor padrão para esta variável. Se e somente se não estiver definido, o script PODE tentar determinar o tipo de mídia a partir dos dados recebidos. Se o tipo permanecer desconhecido, então o script PODE optar por assumir um tipo de `aplicação/fluxo de octetos` ou pode rejeitar a solicitação com erro (conforme descrito na seção 6.3.3).

Cada tipo de mídia define um conjunto de parâmetros opcionais e obrigatórios. Isso pode incluir um parâmetro charset com um valor que não diferencia maiúsculas de minúsculas, definindo o conjunto de caracteres codificados para o corpo da mensagem. Se o parâmetro charset for omitido, o valor padrão deverá ser derivado de acordo com qualquer uma das seguintes regras a ser aplicada primeiro:

1. PODE haver um conjunto de caracteres padrão definido pelo sistema para alguns tipos de mídia.
2. O padrão para tipos de mídia do tipo “texto” é ISO-8859-1 [4].
3. Qualquer padrão definido na especificação do tipo de mídia.
4. O padrão é US-ASCII.

O servidor DEVE definir esta metavariável se um campo HTTP Content-Type estiver presente no cabeçalho da solicitação do cliente. Se o servidor receber uma solicitação com uma entidade anexada, mas sem campo de cabeçalho Content-Type, ele PODE tentar determinar o tipo de conteúdo correto, caso contrário, deverá omitir

esta metavariável.

4.1.4 GATEWAY_INTERFACE

A variável GATEWAY_INTERFACE DEVE ser configurada para o dialeto do CGI que está sendo usado pelo servidor para se comunicar com o script. Sintaxe:

```
GATEWAY_INTERFACE = "CGI" "/" 1*dígito "." 1 * dígito
```

Observe que os números maiores e menores são tratados como números inteiros separados e, portanto, cada um pode ser incrementado acima de um único dígito. Portanto, CGI/2.4 é uma versão inferior ao CGI/2.13, que por sua vez é inferior ao CGI/12.3. Zeros à esquerda DEVEM ser ignorados pelo script e NÃO DEVEM ser gerados pelo servidor.

Este documento define a versão 1.1 da interface CGI.

4.1.5 PATH_INFO

A variável PATH_INFO especifica um caminho a ser interpretado pelo script CGI. Ele identifica o recurso ou sub-recurso a ser retornado pelo script CGI e é derivado da parte da hierarquia do caminho URI que segue a parte que identifica o próprio script. Ao contrário de um caminho URI, PATH_INFO não é codificado por URL e não pode conter parâmetros de segmento de caminho. Um PATH_INFO de "/" representa um único segmento de caminho vazio.

```
PATH_INFO  =  "" | ( "/" caminho )
caminho    =  lsegmento *( "/" lsegmento )
segmento   =  *Risos
rindo      =  <qualquer TEXTO ou CTL
              exceto "/">
```

O valor é considerado sensível a maiúsculas e minúsculas e o servidor DEVE preservar a capitalização do caminho conforme apresentado no URI da solicitação. O servidor PODE impor restrições e limitações sobre quais valores permite para PATH_INFO e PODE rejeitar a solicitação com um erro se encontrar quaisquer valores considerados questionáveis. Isso PODE incluir quaisquer solicitações que resultariam na decodificação de um "/" codificado em PATH_INFO, pois isso pode representar uma perda de informações para o script. Da mesma forma, o tratamento de caracteres não US-ASCII no caminho é definido pelo sistema.

Codificada em URL, a string PATH_INFO forma o componente de caminho extra do Script-URI (consulte a seção 3.3) que segue a parte SCRIPT_NAME desse caminho.

4.1.6 PATH_TRANSLATED

A variável PATH_TRANSLATED é derivada tomando o valor PATH_INFO, analisando-o como um URI local por si só e executando qualquer tradução virtual para física apropriada para mapeá-lo na estrutura de repositório de documentos do servidor. O conjunto de caracteres permitido no resultado é definido pelo sistema.

```
PATH_TRANSLATED = *<qualquer caractere>
```

Este é o local do arquivo que seria acessado por uma solicitação de

```
<esquema> "://" <nome do servidor> ":" <porta do servidor> <caminho
extra>
```

onde <scheme> é o esquema para a solicitação original do cliente e <extra-path> é uma versão codificada em URL de PATH_INFO, com ";", "=" e "?" reservado. Por exemplo, uma solicitação como a seguinte:

```
http://somehost.com/cgi-bin/somescript/this%2eis%2epath%3binfo
```

resultaria em um valor PATH_INFO de

```
/este.é.o.caminho;info
```

Um URI interno é construído a partir do esquema, localização do servidor e PATH_INFO codificado em URL:

```
http://somehost.com/this.is.the.path%3binfo
```

Isso seria então traduzido para um local no repositório de documentos do servidor, talvez um caminho de sistema de arquivos mais ou menos assim:

```
/usr/local/www/htdocs/this.is.the.path;info
```

O valor de PATH_TRANSLATED é o resultado da tradução.

O valor é derivado desta forma, independentemente de ser mapeado para um local de repositório válido. O servidor DEVE preservar a caixa do segmento de caminho extra, a menos que o repositório subjacente suporte nomes que não diferenciam maiúsculas de minúsculas. Se o repositório apenas reconhece maiúsculas de minúsculas, preserva maiúsculas de minúsculas ou é cego em relação a nomes de documentos, o servidor não é obrigado a preservar a maiúsculas e minúsculas do segmento original durante a tradução.

O algoritmo de tradução que o servidor usa para derivar PATH_TRANSLATED é definido pela implementação; Os scripts CGI que usam esta variável podem sofrer portabilidade limitada.

O servidor DEVE definir esta metavariável se o URI da solicitação incluir um componente path-info. Se PATH_INFO for NULL, então a variável PATH_TRANSLATED DEVE ser definida como NULL (ou não definida).

4.1.7 QUERY_STRING

A variável QUERY_STRING contém uma pesquisa codificada em URL ou uma sequência de parâmetros; ele fornece informações ao script CGI para afetar ou refinar o documento a ser retornado pelo script.

A sintaxe da URL para uma string de pesquisa é descrita na seção 3 da RFC 2396 [2]. O valor QUERY_STRING diferencia maiúsculas de minúsculas.

```
QUERY_STRING = string de
consulta string de consulta
= *uric
úrico         = reservado | sem reservas | escapou
```

Ao analisar e decodificar a string de consulta, os detalhes da análise, os caracteres reservados e o suporte para caracteres não US-ASCII dependem do contexto. Por exemplo, o envio de um formulário a partir de um documento HTML [18] usa a codificação application/x-www-form-urlencoded, na qual os caracteres "+", "&" e

“=” são reservados, e a codificação ISO 8859-1 pode ser usado para caracteres não US-ASCII.

O valor QUERY_STRING fornece a parte da string de consulta do Script-URI. (Ver seção 3.3).

O servidor DEVE definir esta variável; se o Script-URI não incluir um componente de consulta, QUERY_STRING DEVE ser definido como uma string vazia (“”).

4.1.8 REMOTE_ADDR

A variável REMOTE_ADDR DEVE ser definida como o endereço de rede do cliente que envia a solicitação ao servidor.

```
REMOTE_ADDR    =  número do host
número do      =  endereço IPv4 | endereço ipv6
host
endereço ipv4  =  1*3 dígitos "." 1*3 dígitos "." 1*3 dígitos "."
                  1*3 dígitos
endereço ipv6  =  hexpart [ ":" endereço ipv4]
parte          =  hexseq | ([ hexseq ] "::" [ hexseq ] )
hexadecimal
hexseq         =  1*4 hexadecimal *( ":" 1*4 hexadecimal)
```

O formato de um endereço IPv6 é descrito na RFC 3513 [15].

4.1.9 REMOTE_HOST

A variável REMOTE_HOST contém o nome de domínio totalmente qualificado do cliente que envia a solicitação ao servidor, se disponível, caso contrário, NULL. Nomes de domínio totalmente qualificados assumem a forma descrita na seção 3.5 da RFC 1034 [17] e na seção 2.1 da RFC 1123 [12]. Os nomes de domínio não diferenciam maiúsculas de minúsculas.

```
REMOTE_HOST    =  "" | nome do host | número do host
nome do host   =  *( domainlabel "." ) toplabel [ "." ]
rótulo de      =  alfanum [ *alfahipdígito alfanum ]
domínio
rótulo         =  alfa [ *alfahipdígito alfanum ]
superior
alfa-hipdígito =  alfanum | "-"
```

O servidor DEVE definir esta variável. Se o nome do host não estiver disponível por motivos de desempenho ou não, o servidor PODE substituir o valor REMOTE_ADDR.

4.1.10 REMOTE_IDENT

A variável REMOTE_IDENT PODE ser usada para fornecer informações de identidade relatadas sobre a conexão por uma solicitação RFC 1413 [20] ao agente remoto, se disponível. O servidor pode optar por não suportar esta funcionalidade, ou não solicitar os dados por razões de eficiência, ou não devolver os dados de identidade disponíveis.

```
REMOTE_IDENT = *TEXTO
```

Os dados retornados podem ser usados para fins de autenticação, mas o nível de confiança neles depositado deve ser mínimo.

4.1.11 REMOTE_USER

A variável REMOTE_USER fornece uma string de identificação do usuário fornecida pelo cliente como parte da autenticação do usuário.

```
REMOTE_USER = *TEXTO
```

Se a solicitação do cliente exigir autenticação HTTP [5] (por exemplo, a metavariável AUTH_TYPE está definida como “Básico” ou “Digest”), então o valor da metavariável REMOTE_USER DEVE ser definido como o ID do usuário fornecido.

4.1.12 REQUEST_METHOD

A metavariável REQUEST_METHOD DEVE ser definida como o método que deve ser usado pelo script para processar a solicitação, conforme descrito na seção 4.3.

```
REQUEST_METHOD    = método
método            = "OBTER" | "POSTAR" | "CABEÇA" | método de
extensão método de extensão = "PUT" | "EXCLUIR" | ficha
```

O método diferencia maiúsculas de minúsculas. Os métodos HTTP são descritos na seção 5.1.1 da especificação HTTP/1.0 [1] e na seção 5.1.1 da especificação HTTP/1.1 [4].

4.1.13 SCRIPT_NAME

A variável SCRIPT_NAME DEVE ser definida como um caminho URI (não codificado em URL) que possa identificar o script CGI (em vez da saída do script). A sintaxe é a mesma de PATH_INFO (seção 4.1.5)

```
SCRIPT_NAME = "" | ( "/" caminho )
```

O “/” inicial não faz parte do caminho. É opcional se o caminho for NULL; no entanto, a variável ainda DEVE ser definida nesse caso.

A string SCRIPT_NAME forma uma parte principal do componente de caminho do Script-URI derivado de alguma maneira definida pela implementação. Nenhum segmento PATH_INFO (consulte a seção 4.1.5) está incluído no valor SCRIPT_NAME.

4.1.14 SERVER_NAME

A variável SERVER_NAME DEVE ser definida como o nome do host do servidor para o qual a solicitação do cliente é direcionada. É um nome de host ou endereço de rede que não diferencia maiúsculas de minúsculas. Ele forma a parte host do Script-URI.

```
SERVER_NAME = nome do servidor
nome do servidor = nome do host | endereço IPv4 | ( "[" endereço ipv6 "]"
```

Um servidor implantado pode ter mais de um valor possível para esta variável, onde vários hosts virtuais HTTP compartilham

o mesmo endereço IP. Nesse caso, o servidor usaria o conteúdo do campo de cabeçalho Host da solicitação para selecionar o host virtual correto.

4.1.15 SERVER_PORT

A variável SERVER_PORT DEVE ser definida como o número da porta TCP/IP na qual esta solicitação é recebida do cliente. Este valor é usado na parte port do Script-URI.

```
SERVER_PORT = porta do
servidor porta do servidor
= 1*dígito
```

Observe que esta variável DEVE ser definida, mesmo que a porta seja a porta padrão para o esquema e possa ser omitida de um URI.

4.1.16 SERVER_PROTOCOL

A variável SERVER_PROTOCOL DEVE ser definida com o nome e a versão do protocolo de aplicação usado para esta solicitação CGI. Isto PODE diferir da versão do protocolo utilizada pelo servidor na sua comunicação com o cliente.

```
SERVER_PROTOCOL    =  Versão HTTP | "INCLUÍDO" | versão de extensão
Versão HTTP        =  "HTTP" "/" 1*dígito "." 1 * dígito
versão de extensão =  protocolo [ "/" 1*dígito "." 1*dígito]
protocolo          =  ficha
```

Aqui, ‘protocolo’ define a sintaxe de algumas das informações que passam entre o servidor e o script (os recursos ‘específicos do protocolo’). Não faz distinção entre maiúsculas e minúsculas e geralmente é apresentado em letras maiúsculas. O protocolo não é o mesmo que a parte do esquema do URI do script, que define o mecanismo geral de acesso usado pelo cliente para se comunicar com o servidor. Por exemplo, uma solicitação que chega ao script com um protocolo “HTTP” pode ter usado um esquema “https”.

Um valor bem conhecido para SERVER_PROTOCOL que o servidor PODE usar é “INCLUDED”, que sinaliza que o documento atual está sendo incluído como parte de um documento composto, em vez de ser o alvo direto da solicitação do cliente. O script deve tratar isso como uma solicitação HTTP/1.0.

4.1.17 SERVIDOR_SOFTWARE

A metavariável SERVER_SOFTWARE DEVE ser definida como o nome e a versão do software do servidor de informações que faz a solicitação CGI (e executa o gateway). DEVE ser igual à descrição do servidor informada ao cliente, se houver.

```
SERVER_SOFTWARE = 1*(produto | comentário)
produto          = token [ "/" versão do produto]
```

```
versão do produto   =  ficha
comentário          =  "(" *( ctext | comentário ) ")"
texto               =  <qualquer TEXTO excluindo "(" e
                       ")">
```

4.1.18 Metavariáveis específicas do protocolo

O servidor DEVE definir meta-variáveis específicas para o protocolo e esquema para a solicitação. A interpretação das variáveis específicas do protocolo depende da versão do protocolo em SERVER_PROTOCOL. O servidor PODE definir uma metavariável com o nome do esquema para um valor não NULO se o esquema não for igual ao protocolo. A presença de tal variável indica ao script qual esquema é utilizado pela solicitação.

Metavariáveis com nomes iniciados por “HTTP_” contêm valores lidos dos campos do cabeçalho da solicitação do cliente, se o protocolo utilizado for HTTP. O nome do campo do cabeçalho HTTP é convertido em letras maiúsculas, tem todas as ocorrências de “-” substituídas por “_” e tem “HTTP_” anexado para fornecer o nome da metavariável. Os dados do cabeçalho podem ser apresentados como enviados pelo cliente ou podem ser reescritos de forma que não alterem sua semântica. Se vários campos de cabeçalho com o mesmo nome de campo forem recebidos, o servidor DEVE reescrevê-los como um único valor com a mesma semântica. Da mesma forma, um campo de cabeçalho que abrange várias linhas DEVE ser mesclado em uma única linha. O servidor DEVE, se necessário, alterar a representação dos dados (por exemplo, o conjunto de caracteres) para ser apropriado para uma metavariável CGI.

O servidor não é obrigado a criar metavariáveis para todos os campos de cabeçalho que recebe. Em particular, DEVE remover quaisquer campos de cabeçalho que contenham informações de autenticação, como ‘Autorização’; ou que estão disponíveis para o script em outras variáveis, como ‘Content-Length’ e ‘Content-Type’. O servidor PODE remover campos de cabeçalho relacionados exclusivamente a problemas de comunicação do lado do cliente, como ‘Conexão’.

4.2 Solicitar corpo da mensagem

Os dados da solicitação são acessados pelo script em um método definido pelo sistema; a menos que definido de outra forma, isso será feito lendo o descritor de arquivo ou identificador de arquivo de ‘entrada padrão’.

```
Dados de solicitação      = [corpo da solicitação]
[dados de extensão] corpo da solicitação    =
<CONTENT_LENGTH>OCTETO
dados de extensão = *OCTETO
```

Um corpo de solicitação é fornecido com a solicitação se CONTENT_LENGTH não for NULL. O servidor DEVE disponibilizar pelo menos essa quantidade de bytes para leitura do script. O servidor PODE sinalizar uma condição de fim de arquivo após a leitura dos bytes CONTENT_LENGTH ou PODE fornecer dados de extensão. Portanto, o script NÃO DEVE tentar ler mais do que CONTENT_LENGTH bytes, mesmo que haja mais dados disponíveis. No entanto, não é obrigado a ler nenhum dos dados.

Para scripts de cabeçalho não analisado (NPH) (seção 5), o servidor DEVE tentar garantir que os dados fornecidos ao script sejam exatamente os fornecidos pelo cliente e inalterados pelo servidor.

Como as codificações de transferência não são suportadas no corpo da solicitação, o servidor DEVE remover tais codificações do

corpo da mensagem e recalcular CONTENT_LENGTH. Se isso não for possível (por exemplo, devido a grandes requisitos de buffer), o servidor DEVE rejeitar a solicitação do cliente. Também PODE remover codificações de conteúdo do corpo da mensagem.

4.3 Métodos de solicitação

O Método de Solicitação, conforme fornecido na metavariável REQUEST_METHOD, identifica o método de processamento a ser aplicado pelo script na produção de uma resposta. O autor do script pode optar por implementar os métodos mais apropriados para a aplicação específica. Se o script receber uma solicitação com um método que não suporta, DEVE rejeitá-lo com um erro (ver seção 6.3.3).

4.3.1 PEGAR

O método GET indica que o script deve produzir um documento baseado nos valores da metavariável. Por convenção, o método GET é “seguro” e “idempotente” e NÃO DEVE ter o significado de realizar outra ação além de produzir um documento.

O significado do método GET pode ser modificado e refinado por metavariáveis específicas do protocolo.

4.3.2 PUBLICAR

O método POST é utilizado para solicitar que o script execute o processamento e produza um documento com base nos dados do corpo da mensagem da solicitação, além dos valores das metavariáveis. Um uso comum é o envio de formulários em HTML [18], destinado a iniciar o processamento pelo script que tem efeito permanente, como uma alteração em um banco de dados.

O script DEVE verificar o valor da variável CONTENT_LENGTH antes de ler o corpo da mensagem anexado, e DEVE verificar o valor de CONTENT_TYPE antes de processá-lo.

4.3.3 CABEÇA

O método HEAD solicita que o script faça processamento suficiente para retornar os campos do cabeçalho de resposta, sem fornecer um corpo de mensagem de resposta. O script NÃO DEVE fornecer um corpo de mensagem de resposta para uma solicitação HEAD. Se isso acontecer, o servidor DEVE descartar o corpo da mensagem ao ler a resposta do script.

4.3.4 Métodos Específicos de Protocolo

O script PODE implementar qualquer método específico de protocolo, como HTTP/1.1 PUT e DELETE; DEVE verificar o valor de SERVER_PROTOCOL ao fazer isso.

O servidor PODE decidir que alguns métodos não são apropriados ou permitidos para um script e pode manipular os métodos sozinho ou retornar um erro ao cliente.

4.4 A linha de comando do script

Alguns sistemas suportam um método para fornecer uma matriz de strings ao script CGI. Isso é usado apenas no caso de uma consulta HTTP 'indexada', que é identificada por uma solicitação 'GET' ou 'HEAD' com uma string de consulta URI que não contém nenhum caractere "=" não codificado. Para tal solicitação, o servidor DEVE tratar a string de consulta como uma string de pesquisa e analisá-la em palavras, usando as regras

```
string de pesquisa  = palavra de pesquisa *( "+" palavra de
                    = pesquisa) 1*schar
palavra de pesquisa = sem reservas | escapou | xreservado
schar
xreservado          = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "," |
                      "$"
```

Após a análise, cada palavra de pesquisa é decodificada por URL, opcionalmente codificada de uma maneira definida pelo sistema e depois adicionada à lista de argumentos da linha de comando.

Se o servidor não puder criar nenhuma parte da lista de argumentos, então o servidor NÃO DEVE gerar nenhuma informação de linha de comando. Por exemplo, o número de argumentos pode ser maior que os limites do sistema operacional ou do servidor, ou uma das palavras pode não ser representável como argumento.

O script DEVE verificar se o valor QUERY_STRING contém um caractere "=" não codificado e NÃO DEVE usar os argumentos da linha de comando, se isso acontecer.

5 Scripts da CNP

5.1 Identificação

O servidor PODE suportar scripts NPH (Non-Parsed Header); estes são scripts para os quais o servidor passa toda a responsabilidade pelo processamento da resposta.

Esta especificação não fornece nenhum mecanismo para que um script NPH seja identificado apenas com base em seus dados de saída. Por convenção, portanto, qualquer script específico só pode fornecer saída de um tipo (NPH ou CGI) e, portanto, o script em si é descrito como um 'script NPH'. Um servidor com suporte NPH DEVE fornecer um mecanismo definido pela implementação para identificar scripts NPH, talvez com base no nome ou localização do script.

5.2 Resposta da CNP

DEVE haver um método definido pelo sistema para o script enviar dados de volta ao servidor ou cliente; um script DEVE sempre retornar alguns dados. A menos que definido de outra forma, será igual aos scripts CGI convencionais.

Atualmente, os scripts NPH são definidos apenas para solicitações de clientes HTTP. Um script NPH (HTTP) DEVE retornar uma mensagem de resposta HTTP completa, atualmente descrita na seção 6 das especificações HTTP [1], [4]. O script DEVE usar a variável SERVER_PROTOCOL para determinar o formato apropriado para uma resposta. Deve também ter em conta quaisquer meta-variáveis genéricas ou específicas do protocolo no pedido, conforme possa ser exigido pela especificação do protocolo específico.

O servidor DEVE garantir que a saída do script seja enviada ao cliente sem modificações. Observe que isso requer que o script use o conjunto de caracteres correto (US-ASCII [9] e ISO 8859-1 [10] para HTTP) nos campos de cabeçalho. O servidor DEVE tentar garantir que a saída do script seja enviada diretamente ao cliente, com buffer interno mínimo e sem buffer visível para transporte.

A menos que a implementação defina o contrário, o script NÃO DEVE indicar em sua resposta que o cliente pode enviar mais solicitações pela mesma conexão.

6 Resposta CGI

6.1 Tratamento de respostas

Um script DEVE sempre fornecer uma resposta não vazia e, portanto, existe um método definido pelo sistema para enviar esses dados de volta ao servidor. A menos que definido de outra forma, isso será feito por meio do descritor de arquivo ‘saída padrão’.

O script DEVE verificar a variável REQUEST_METHOD ao processar a solicitação e preparar sua resposta.

O servidor PODE implementar um período de tempo limite dentro do qual os dados devem ser recebidos do script. Se uma implementação de servidor definir esse tempo limite e não receber dados de um script dentro do período de tempo limite, o servidor PODE encerrar o processo de script.

6.2 Tipos de resposta

A resposta compreende um cabeçalho e um corpo da mensagem, separados por uma linha em branco. O cabeçalho da mensagem contém um ou mais campos de cabeçalho. O corpo pode ser NULL.

```
resposta genérica = 1*campo de cabeçalho NL [corpo de resposta]
```

O script DEVE retornar uma resposta de documento, uma resposta de redirecionamento local ou uma resposta de redirecionamento de cliente (com documento opcional). Nas definições de resposta abaixo, a ordem dos campos de cabeçalho em uma resposta não é significativa (apesar de aparecer assim na BNF). Os campos de

cabeçalho são definidos na seção 6.3.

```
Resposta CGI = resposta ao documento | resposta de
                redirecionamento local | resposta de
                redirecionamento do cliente | resposta do
                cliente-redirdoc
```

6.2.1 Resposta do Documento

O script CGI pode retornar um documento ao usuário em uma resposta de documento, com um código de erro opcional indicando o status de sucesso da resposta.

```
resposta do documento = Tipo de conteúdo [Status] *outro campo NL
                    corpo de resposta
```

O script DEVE retornar um campo de cabeçalho Content-Type. Um campo de cabeçalho Status é opcional e o status 200 ‘OK’ é assumido se for omitido. O servidor DEVE fazer quaisquer modificações apropriadas na saída do script para garantir que a resposta ao cliente esteja em conformidade com a versão do protocolo de resposta.

6.2.2 Resposta de redirecionamento local

O script CGI pode retornar um caminho URI e uma string de consulta (“local-pathquery”) para um recurso local em um cabeçalho Location
campo. Isto indica ao servidor que ele deve reprocessar a solicitação usando o caminho especificado.

```
resposta-redirecionamento local = local-local NL
```

O script NÃO DEVE retornar nenhum outro campo de cabeçalho ou corpo da mensagem, e o servidor DEVE gerar a resposta que teria produzido em resposta a uma solicitação contendo a URL

```
esquema "://" nome do servidor ":" porta do servidor
                local-pathquery
```

6.2.3 Resposta de redirecionamento do cliente

O script CGI pode retornar um caminho URI absoluto em um campo de cabeçalho Location, para indicar ao cliente que ele deve reprocessar a solicitação usando o URI especificado.

```
cliente-redir-resposta = localização do cliente *campo de extensão NL
```

O script NÃO DEVE fornecer quaisquer outros campos de cabeçalho, exceto campos de extensão CGI definidos pelo servidor. Para uma solicitação de cliente HTTP, o servidor DEVE gerar uma mensagem de resposta HTTP 302 ‘Encontrada’.

6.2.4 Resposta de redirecionamento do cliente com documento

O script CGI pode retornar um caminho URI absoluto em um campo de cabeçalho Location junto com um documento anexado, para indicar ao cliente que ele deve reprocessar a solicitação usando o URI especificado.

```
client-redirdoc-response = tipo de conteúdo do status do local do cliente
                           *corpo de resposta NL de outro campo
```

O campo do cabeçalho Status DEVE ser fornecido e DEVE conter um valor de status de 302 ‘Encontrado’, ou PODE conter um código de extensão, ou seja, outro código de status válido que significa redirecionamento do cliente. O servidor DEVE fazer quaisquer modificações apropriadas na saída do script para garantir que a resposta ao cliente esteja em conformidade com a versão do protocolo de resposta.

6.3 Campos de cabeçalho de resposta

Os campos de cabeçalho de resposta são campos CGI ou de cabeçalho de extensão a serem interpretados pelo servidor, ou campos de cabeçalho específicos do protocolo a serem incluídos na resposta retornada ao cliente. Pelo menos um campo CGI DEVE ser fornecido; cada campo CGI NÃO DEVE aparecer mais de uma vez na resposta. Os campos do cabeçalho de resposta possuem a sintaxe:

```
campo de cabeçalho  =  Campo CGI | outro campo
Campo CGI           =  Tipo de conteúdo | Localização | Status
outro campo         =  campo de protocolo | campo de extensão
campo de protocolo  =  campo genérico
campo de extensão   =  campo genérico
campo genérico      =  nome do campo ":" [valor do campo] NL
nome do campo       =  ficha
valor do campo      =  *( conteúdo do campo | LWSP )
conteúdo do campo   =  *( token | separador | string entre
                       aspas )
```

O nome do campo não diferencia maiúsculas de minúsculas. Um valor de campo NULL equivale a um campo que não está sendo enviado. Observe que cada campo de cabeçalho em uma resposta CGI DEVE ser especificado em uma única linha; CGI/1.1 não suporta linhas de continuação. Espaços em branco são permitidos entre “:” e o valor do campo (mas não entre o nome do campo e “:”), e também entre tokens no valor do campo.

6.3.1 Tipo de conteúdo

O campo de resposta Content-Type define o Internet Media Type [6] do corpo da entidade.

```
Content-Type = "Content-Type:" tipo de mídia NL
```

Se um corpo de entidade for retornado, o script DEVE fornecer um campo Content-Type na resposta. Se isso não acontecer, o servidor NÃO DEVE tentar determinar o tipo de conteúdo correto. O valor DEVE ser enviado sem modificação ao cliente, exceto para quaisquer alterações nos parâmetros do charset.

A menos que seja definido de outra forma pelo sistema, o conjunto de caracteres padrão assumido pelo cliente para tipos de mídia de texto é ISO-8859-1 se o protocolo for HTTP e US-ASCII caso contrário. Portanto, o

script DEVE incluir um parâmetro charset. Consulte a seção 3.4.1 da especificação HTTP/1.1 [4] para uma discussão sobre esse problema.

6.3.2 Localização

O campo de cabeçalho Location é usado para especificar ao servidor que o script está retornando uma referência a um documento em vez de um documento real (consulte as seções 6.2.3 e 6.2.4). É um URI absoluto (opcionalmente com um identificador de fragmento), indicando que o cliente deve buscar o documento referenciado, ou um caminho de URI local (opcionalmente com uma string de consulta), indicando que o servidor deve buscar o documento referenciado e retornar para o cliente como resposta.

```
Localização      =  local-Localização | localização do
                    cliente
localização do   =  "Localização:" fragmento-URI NL
cliente
local-Localizaçã =  "Localização:" local-pathquery NL
o
fragmento-URI    =  absolutoURI [fragmento "#"]
fragmento        =  *úrico
consulta de      =  caminho abs [ "?" string de consulta]
caminho local
caminho          =  "/" segmentos de caminho
abdominal
segmentos de     =  segmento *( segmento "/" )
caminho
segmento         =  *pchar
pchar            =  sem reservas | escapou | extra
extra            =  ":" | "@" | "&" | "=" | "+" | "$" | ","
```

A sintaxe de um absolutoURI é incorporada neste documento a partir daquela especificada na RFC 2396 [2] e na RFC 2732 [7]. Um absolutoURI válido sempre começa com o nome do esquema seguido de “:”; os nomes dos esquemas começam com uma letra e continuam com caracteres alfanuméricos, “+”, “-” ou “.”. O caminho do URI local e a consulta devem ser um caminho absoluto e não um caminho relativo ou NULL e, portanto, devem começar com “/”.

Observe que qualquer corpo de mensagem anexado à solicitação (como uma solicitação POST) pode não estar disponível para o recurso que é o destino do redirecionamento.

6.3.3 Status

O campo de cabeçalho Status contém um código de resultado inteiro de 3 dígitos que indica o nível de sucesso da tentativa do script de lidar com a solicitação.

```
Status          = "Status:" código de status SP frase de razão NL
código de status= "200" | "302" | "400" | "501" | código de
extensão código de extensão = 3 dígitos
frase-razão     = *TEXTO
```

O código de status 200 ‘OK’ indica sucesso e é o valor padrão assumido para uma resposta de documento. O código de status 302 ‘Encontrado’ é usado com um campo de cabeçalho de localização e corpo da mensagem de resposta. O código de status 400 ‘Solicitação incorreta’ pode ser usado para um formato de solicitação desconhecido, como CONTENT_TYPE ausente. O código de status 501 ‘Not Implemented’ pode ser retornado

por um script se receber um REQUEST_METHOD não suportado.

Outros códigos de status válidos estão listados na seção 6.1.1 das especificações HTTP [1], [4] e também no Registro de códigos de status HTTP da IANA [8] e PODEM ser usados em adição ou em vez dos listados acima. O roteiro DEVE

verifique o valor de SERVER_PROTOCOL antes de usar códigos de status HTTP/1.1. O script PODE rejeitar com o erro 405 ‘Método não permitido’ solicitações HTTP/1.1 feitas usando um método que não suporta.

Observe que retornar um código de status de erro não significa necessariamente uma condição de erro no próprio script. Por exemplo, um script invocado como manipulador de erros pelo servidor deve retornar o código apropriado à condição de erro do servidor.

A frase-razão é uma descrição textual do erro a ser devolvida ao cliente para consumo humano.

6.3.4 Campos de cabeçalho específicos do protocolo

O script PODE retornar quaisquer outros campos de cabeçalho relacionados à mensagem de resposta definida pela especificação do SERVER_PROTOCOL (HTTP/1.0 [1] ou HTTP/1.1 [4]). O servidor DEVE traduzir os dados do cabeçalho da sintaxe do cabeçalho CGI para a sintaxe do cabeçalho HTTP, se estes diferirem. Por exemplo, a sequência de caracteres para nova linha (como US-ASCII LF do UNIX) usada por scripts CGI pode não ser a mesma usada por HTTP (US-ASCII CR seguido de LF).

O script NÃO DEVE retornar nenhum campo de cabeçalho relacionado a problemas de comunicação do lado do cliente e que possa afetar a capacidade do servidor de enviar a resposta ao cliente. O servidor PODE remover quaisquer campos de cabeçalho retornados pelo cliente. Ele DEVE resolver quaisquer conflitos entre os campos de cabeçalho retornados pelo script e os campos de cabeçalho que, de outra forma, ele próprio enviaria.

6.3.5 Campos de cabeçalho de extensão

Pode haver campos de cabeçalho CGI adicionais definidos pela implementação, cujos nomes de campo DEVEM começar com “X-CGI-”. O servidor PODE ignorar (e excluir) quaisquer campos de cabeçalho não reconhecidos com nomes começando com “X-CGI-” recebidos do script.

6.4 Corpo da mensagem de resposta

O corpo da mensagem de resposta é um documento anexado a ser retornado ao cliente pelo servidor. O servidor DEVE ler todos os dados fornecidos pelo script, até que o script sinalize o fim do corpo da mensagem por meio de uma condição de fim de arquivo. O corpo da mensagem DEVE ser enviado sem modificações ao cliente, exceto para solicitações HEAD ou quaisquer codificações de transferência, codificações de conteúdo ou conversões de conjunto de caracteres necessárias.

```
corpo de resposta = *OCTETO
```

7 Especificações do sistema

7.1 AmigaDOS

Metavariáveis
Metavariáveis são passadas para o script em variáveis de ambiente com nomes idênticos. Eles são acessados pela rotina da biblioteca DOS `ObterVar()`. O `bandeiras` argumento DEVE ser `0`. As maiúsculas e minúsculas são ignoradas, mas as maiúsculas são recomendadas para compatibilidade com sistemas que diferenciam maiúsculas de minúsculas.

O diretório de trabalho atual
O diretório de trabalho atual do script é definido como o diretório que contém o script.

Conjunto de caracteres
O conjunto de caracteres US-ASCII [9] é usado para a definição de metavariáveis, campos de cabeçalho e valores; a sequência de nova linha (NL) é LF; os servidores também DEVEM aceitar CR LF como nova linha.

7.2 UNIX

Para sistemas operacionais compatíveis com UNIX, são definidos os seguintes:

Metavariáveis
Metavariáveis são passadas para o script em variáveis de ambiente com nomes idênticos. Eles são acessados pela rotina da biblioteca C `getenv()` ou variável `aproximadamente`.

A linha de comando
Isso é acessado usando o `argc` e `argumento` argumentos para `principal()`. As palavras têm quaisquer caracteres que estejam “ativos” no shell Bourne escapados com uma barra invertida.

O diretório de trabalho atual
O diretório de trabalho atual do script DEVE ser definido como o diretório que contém o script.

Conjunto de caracteres
O conjunto de caracteres US-ASCII [9], excluindo NUL, é usado para a definição de metavariáveis, campos de cabeçalho e valores CHAR; Os valores de TEXTO usam ISO-8859-1. O valor PATH_TRANSLATED pode conter qualquer byte de 8 bits, exceto NUL. A sequência de nova linha (NL) é LF; os servidores também devem aceitar CR LF como nova linha.

7.3 EBCDIC/POSIX

Para sistemas operacionais compatíveis com POSIX que usam o conjunto de caracteres EBCDIC, são definidos os seguintes:

Metavariáveis

Metavariáveis são passadas para o script em variáveis de ambiente com nomes idênticos. Eles são acessados pela rotina da biblioteca C `getenv()`.

A linha de comando
Isso é acessado usando o `argc` e `argumento` argumentos para `principal()`. As palavras têm quaisquer caracteres que estejam “ativos” no shell Bourne escapados com uma barra invertida.

O diretório de trabalho atual
O diretório de trabalho atual do script DEVE ser definido como o diretório que contém o script.

Conjunto de caracteres
O conjunto de caracteres IBM1047 [21], excluindo NUL, é usado para a definição de metavariáveis, campos de cabeçalho, valores, strings TEXT e o valor PATH_TRANSLATED. A sequência de nova linha (NL) é LF; os servidores também devem aceitar CR LF como nova linha.

padrão do conjunto de caracteres do tipo de mídia
O valor do conjunto de caracteres padrão para tipos de mídia de texto (e outros definidos pela implementação) é IBM1047.

8 Implementação

8.1 Recomendações para servidores

Embora o servidor e o script CGI não precisem ser consistentes no tratamento dos caminhos de URL (URLs do cliente e os dados PATH_INFO, respectivamente), os autores do servidor podem desejar impor consistência. Portanto a implementação do servidor deverá especificar seu comportamento para os seguintes casos:

1. definir quaisquer restrições sobre segmentos de caminho permitidos, em particular se segmentos NULL não terminais são permitidos;
2. definir o comportamento para `"."` ou `".."` segmentos de caminho; isto é, se eles são proibidos, tratados como segmentos de caminho comuns ou interpretados de acordo com a especificação de URL relativa [2];
3. defina quaisquer limites da implementação, incluindo limites no comprimento do caminho ou da string de pesquisa e limites no volume de campos de cabeçalho que o servidor analisará.

8.2 Recomendações para scripts

Se o script não pretende processar os dados PATH_INFO, ele deverá rejeitar a solicitação com 404 Not Found se PATH_INFO não for NULL.

Se a saída de um formulário estiver sendo processada, verifique se CONTENT_TYPE é “application/x-www-form-urlencoded”
[18] ou “multipart/form-data” [16]. Se CONTENT_TYPE estiver em branco, o script poderá rejeitar a solicitação com um erro 415 ‘Unsupported Media Type’, quando suportado pelo protocolo.

Ao analisar PATH_INFO, PATH_TRANSLATED ou SCRIPT_NAME o script deve ter cuidado com segmentos de caminho vazios (`"//"`) e segmentos de caminho especiais (`"."` e `".."`). Eles devem ser removidos do

caminho antes de serem usados em chamadas do sistema operacional ou a solicitação deve ser rejeitada com 404 ‘Não encontrado’.

Ao retornar campos de cabeçalho, o script deve tentar enviar os campos de cabeçalho CGI o mais rápido possível e enviá-los antes de qualquer campo de cabeçalho HTTP. Isso pode ajudar a reduzir os requisitos de memória do servidor.

Os autores do script devem estar cientes de que as metavariáveis REMOTE_ADDR e REMOTE_HOST (ver seções 4.1.8 e 4.1.9) podem não identificar a origem final da solicitação. Eles identificam o cliente para a solicitação imediata ao servidor; esse cliente pode ser um proxy, gateway ou outro intermediário agindo em nome do cliente de origem real.

9 Considerações de segurança

9.1 Métodos Seguros

Conforme discutido nas considerações de segurança das especificações HTTP [1], [4], foi estabelecida a convenção de que os métodos GET e HEAD devem ser 'seguros' e 'idempotentes' (solicitações repetidas têm o mesmo efeito que uma única solicitação) . Consulte a seção 9.1 da RFC 2616 [4] para uma discussão completa.

9.2 Campos de cabeçalho contendo informações confidenciais

Alguns campos de cabeçalho HTTP podem conter informações confidenciais que o servidor não deve transmitir ao script, a menos que seja explicitamente configurado para fazê-lo. Por exemplo, se o servidor proteger o script usando o esquema de autenticação Básico, o cliente enviará um campo de cabeçalho de Autorização contendo um nome de usuário e uma senha. O servidor valida essas informações e, portanto, não deve transmitir a senha por meio da metavariável HTTP_AUTHORIZATION sem uma consideração cuidadosa. Isso também se aplica ao campo de cabeçalho Proxy-Authorization e à metavariável HTTP_PROXY_AUTHORIZATION correspondente.

9.3 Privacidade de dados

Os dados confidenciais em uma solicitação devem ser colocados no corpo da mensagem como parte de uma solicitação POST e não no URI ou nos cabeçalhos da mensagem. Em alguns sistemas, o ambiente usado para passar metavariáveis para um script pode ser visível para outros scripts ou usuários. Além disso, muitos servidores, proxies e clientes existentes registrarão permanentemente o URI onde ele poderá ficar visível para terceiros.

9.4 Modelo de Segurança da Informação

Para uma conexão de cliente usando TLS, o modelo de segurança se aplica entre o cliente e o servidor, e não entre o cliente e o script. É responsabilidade do servidor lidar com a sessão TLS e, portanto, é o servidor que é autenticado para o cliente, não o script CGI.

Esta especificação não fornece nenhum mecanismo para o script autenticar o servidor que o invocou. Não há integridade imposta nas mensagens de solicitação e resposta CGI.

9.5 Interferência de script com o servidor

A implementação mais comum de CGI invoca o script como um processo filho usando o mesmo usuário e grupo do processo servidor. Deve-se, portanto, garantir que o script não interfira no processo do servidor, na sua configuração, nos documentos ou nos arquivos de log.

Se o script for executado chamando uma função vinculada ao software do servidor (seja em tempo de compilação ou em tempo de execução), devem ser tomadas precauções para proteger a memória central do servidor ou para garantir que código não confiável não possa ser executado.

9.6 Considerações sobre comprimento de dados e buffer

Esta especificação não impõe limites ao comprimento do corpo da mensagem apresentado ao script. O script não deve presumir que buffers alocados estaticamente de qualquer tamanho sejam suficientes para conter todo o envio de uma vez. O uso de um buffer de comprimento fixo sem uma verificação cuidadosa de overflow pode resultar na exploração de vulnerabilidades de “stack-smashing” ou “stack-overflow” do sistema operacional por um invasor. O script pode armazenar grandes envios em disco ou outra mídia de buffer, mas uma sucessão rápida de envios grandes pode resultar em condições de negação de serviço. Se o CONTENT_LENGTH do corpo da mensagem for maior do que as considerações de recursos permitem, os scripts deverão responder com um status de erro apropriado para a versão do protocolo; Os códigos de status potencialmente aplicáveis incluem 503 ‘Serviço indisponível’ (HTTP/1.0 e HTTP/1.1), 413 ‘Entidade de solicitação muito grande’ (HTTP/1.1) e 414 ‘URI de solicitação muito grande’ (HTTP/1.1).

Considerações semelhantes se aplicam ao tratamento da resposta CGI do script pelo servidor. Não há limite para o comprimento do cabeçalho ou do corpo da mensagem retornado pelo script; o servidor não deve presumir que buffers alocados estaticamente de qualquer tamanho sejam suficientes para conter a resposta inteira.

9.7 Processamento sem Estado

A natureza sem estado da Web torna cada execução de script e recuperação de recurso independente de todas as outras, mesmo quando múltiplas solicitações constituem uma única transação conceitual da Web. Por causa disso, um script não deve fazer suposições sobre o contexto do agente do usuário que envia uma solicitação. Em particular, os scripts devem examinar os dados obtidos do cliente e verificar se são válidos, tanto na forma quanto no conteúdo, antes de permitir que sejam usados para fins confidenciais, como entrada para outros aplicativos, comandos ou serviços do sistema operacional. Esses usos incluem (mas não estão limitados a) argumentos de chamada do sistema, gravações de banco de dados, código-fonte avaliado dinamicamente e entrada para faturamento ou outros processos seguros. É importante que os aplicativos sejam protegidos contra entradas inválidas, independentemente de a invalidez ser resultado de erro do usuário, erro lógico ou ação maliciosa.

Os autores de scripts envolvidos em transações com múltiplas solicitações devem ser particularmente cautelosos

ao validar as informações do estado; efeitos indesejáveis podem resultar da substituição de valores perigosos por partes da submissão

que de outra forma poderia ser presumida segura. A subversão deste tipo ocorre quando são feitas alterações nos dados de um estágio anterior da transação que não deveriam ser controlados pelo cliente (por exemplo, elementos de formulário HTML ocultos, cookies, URLs incorporados, etc.).

9.8 Caminhos Relativos

O servidor deve ter cuidado com `".."` segmentos de caminho no URI da solicitação. Eles devem ser removidos ou resolvidos no URI da solicitação antes de serem divididos em caminho de script e caminho extra. Alternativamente, quando o caminho extra é usado para encontrar o PATH_TRANSLATED, deve-se tomar cuidado para evitar que a resolução do caminho forneça caminhos traduzidos fora de uma hierarquia de caminhos esperada.

9.9 Saída de cabeçalho não analisada

Se um script retornar uma saída de cabeçalho não analisada, para ser interpretada pelo cliente em seu protocolo nativo, o script deverá abordar todas as considerações de segurança relacionadas a esse protocolo.

10 Agradecimentos

Este trabalho é baseado na interface CGI original que surgiu das discussões na lista de discussão ‘www-talk’. Em particular, Rob McCool, John Franks, Ari Luotonen, George Phillips e Tony Sanders merecem reconhecimento especial pelos seus esforços na definição e implementação das primeiras versões desta interface.

Este documento também beneficiou enormemente dos comentários e sugestões feitos por Chris Adie, Dave Kristol e Mike Meyer; também David Morris, Jeremy Madea, Patrick McManus, Adam Donahue, Ross Patterson e Harald Alvestrand.

11 Referências

11.1 Referências Normativas

[1] Berners-Lee, T., Fielding, R. e H. Frystyk, “Protocolo de transferência de hipertexto – HTTP/1.0”, RFC 1945, maio de 1996.

[2] Berners-Lee, T., Fielding, R. e L. Masinter, “Uniform Resource Identifiers (URI): Generic Syntax”, RFC 2396, agosto de 1998.

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

[4] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. e T. Berners-Lee, "Protocolo de transferência de hipertexto – HTTP/1.1", RFC 2616, junho 1999.

[5] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A. e L. Stewart, "Autenticação HTTP: Autenticação de acesso básica e resumida", RFC 2617, Junho de 1999.

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

[7] Hinden, R., Carpenter, B. e L. Masinter, "Formato para endereços IPv6 literais em URLs", RFC 2732, dezembro de 1999.

[8] "Registro de código de status HTTP", `http://www.iana.org/assignments/http-status-codes`, IANA.

[9] "Sistemas de Informação – Conjuntos de Caracteres Codificados – Código Padrão Americano de 7 bits para Intercâmbio de Informações (ASCII de 7 bits)", ANSI INCITS.4-1986 (R2002).

[10] "Tecnologia da informação – conjuntos de caracteres gráficos codificados de byte único de 8 bits – Parte 1: Alfabeto latino nº 1", ISO/IEC 8859-1:1998.

11.2 Referências Informativas

[11] Berners-Lee, T., "Identificadores de recursos universais na WWW: uma sintaxe unificadora para a expressão de nomes e endereços de objetos na rede conforme usado na World Wide Web", RFC 1630, junho de 1994.

[12] Braden, R., Ed., "Requisitos para hosts da Internet – aplicação e suporte", STD 3, RFC 1123, outubro de 1989.

[13] Crocker, D., "Padrão para o formato de mensagens de texto da Internet ARPA", STD 11, RFC 822, agosto de 1982.

[14] Dierks, T. e C. Allen, "The TLS Protocol Version 1.0", RFC 2246, janeiro de 1999.

[15] Hinden R. e S. Deering, "Arquitetura de endereçamento do protocolo da Internet versão 6 (IPv6)", RFC 3513, abril de 2003.

[16] Masinter, L., "Retornando valores de formulários: multipart/form-data", RFC 2388, agosto de 1998.

[17] Mockapetris, P., "Nomes de Domínio - Conceitos e Instalações", STD 13, RFC 1034, novembro de 1987.

[18] Raggett, D., Le Hors, A. e I. Jacobs, Eds., "Especificação HTML 4.01", Recomendação W3C de dezembro de 1999, `http://www.w3.org/TR/html401/`.

[19] Rescola, E. "HTTP sobre TLS", RFC 2818, maio de 2000.

[20] Johns, M., "Protocolo de Identificação", RFC 1413, fevereiro de 1993.

[21] Manual de referência do IBM National Language Support Volume 2, SE09-8002-01, março de 1990.

[22] "A Interface de Gateway Comum", `http://hoohoo.ncsa.uiuc.edu/cgi/`, NCSA, Universidade

de Illinois.

12 Endereços dos Autores

David Robinson
Fundação Apache Software
E-mail:
drtr@apache.org Ken
A. L. Coar
E-mail da Apache Software

Foundation: coar@apache.org

13 Declaração completa de direitos autorais

Propriedade intelectual

A IETF não toma posição quanto à validade ou escopo de quaisquer Direitos de Propriedade Intelectual ou outros direitos que possam ser reivindicados como pertencentes à implementação ou uso da tecnologia descrita neste documento ou até que ponto qualquer licença sob tais direitos pode ou não estar disponível; nem representa que tenha feito qualquer esforço independente para identificar tais direitos. Informações sobre os procedimentos da ISOC com relação aos direitos nos Documentos ISOC podem ser encontradas em BCP 78 e BCP 79.

Cópias das divulgações de DPI feitas ao Secretariado da IETF e quaisquer garantias de licenças a serem disponibilizadas, ou o resultado de uma tentativa feita para obter uma licença geral ou permissão para o uso de tais direitos de propriedade por implementadores ou usuários desta especificação podem ser obtidas do repositório on-line de DPI da IETF em `http:// www.ietf.org/ipr`.

A IETF convida qualquer parte interessada a chamar a sua atenção para quaisquer direitos autorais, patentes ou pedidos de patentes, ou outros direitos de propriedade que possam abranger tecnologia que possa ser necessária para implementar este padrão. Por favor, envie as informações para a IETF em ietf-ipr@ietf.org.

Reconhecimento

O financiamento para a função de Editor RFC é atualmente fornecido pela Internet Society.