O que é HTTP

É um protocolo de comunicação entre redes de computadores, permitindo enviar e receber dados do servidor e da máquina cliente.

O que são HTTP Messages

As `HTTP MESSAGES` (mensagens HTTP) são principalmente dividias em dois modelos `Request` (Requisição) e `Response` (Resposta). Em ambos os casos eles possuem a seguinte estrutura:

```
    <message line>
    Header: value
    Another-Header: value

    Message body
```

Os `Headers` (cabeçalhos) são pares de chave/valor (como um array associativo). As chaves são *case insensitive* e os valores são *strings*. O mesmo *header* pode ser emitido várias vezes, e nesse caso, os valores são considerados como uma lista; na maioria dos casos, também podem ser expressados concatenados por vírgula(`,`) como delimitadores.

```
    Header1: ['value1', 'value2']
    // ou na maioria dos casos
    Header1: 'value1 , value2'
```

O `message body` (corpo da mensagem) é uma string mas pode ser manipulada pelo servidor e cliente como uma `Stream` para conservar memória e processamento. É principalmente importante quando transmitem arquivos. A linha da mensagem é o que diferencia uma `request` (requisição) de uma `response` (resposta).

A linha da mensagem de uma requisição é chamada de `request line` (linha de requisição) e tem o seguinte formato

```
    METHOD request-target HTTP/VERSION
```

`METHOD` (método) indica a operação requisitada ao servidor, podendo ser: `GET`, `POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS`, `HEAD`, etc.

A `VERSION` (versão) geralmente é *1.0, 1.1* e em alguns browsers já é possível usar a versão *2.0*.

O `request-target` (alvo da requisição) é onde as coisas começam a ficar um pouco mais complexas. Ele pode assumir 4 diferentes formas:

- `origin-form` (form original) que é o **path**(caminho) e a **query string** [se presente] do **URI**

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

- `absolute-form` (form absoluto) que é um **URI absoluto**

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

- `authority-form` (form de autorização) é uma parte de um uri (**user-info**, se presente; **host**; e **port**, se não for padrão).

```
   CONNECT www.example.com:80 HTTP/1.1
```

- `asterisk-form` contém um caractere asterisco *

```
    OPTIONS * HTTP/1.1
    // Por exemplo de receber uma requisição absolut
a, se comportaria da seguinte maneira
    OPTIONS http://www.example.org:8001 HTTP/1.1
    // Se transformaria em:
    OPTIONS * HTTP/1.1
    Host: www.example.org:8001
```

Message Headers (Cabeçalhos das Mensagens)

As mensagens dos cabeçalhos são *case insensitives*. Infelizmente a maiorias das linguagens de programação e bibliotecas, fazem algum tipo de normalização que torna difícil o consumo dessas mensagens. Por exemplo, no PHP temos a variável global `$_SERVER` que transforma tudo em caixa alta e prefixa as chaves com **HTTP_** e substitui os `_` por `-`.

A **PSR-7** simplifica o acesso aos cabeçalhos provendo uma camada orientada a objetos em cima desses padrões.

```
    // Retorna um array vazio se não encontrar:
    $header = $message->getHeader('Accept');

    // Retorna uma string vazia se não encontrar:
    $header = $message->getHeaderLine('Accept');

    // Verifica se existe um cabeçalho:
    if (! $message->hasHeader('Accept')) { ... }

    // Se existir múltiplos valores, retorna eles como arra
y:
    $values = $message->getHeader('X-Foo');
    // output: array(2)  [ 'value1', 'value2'];

    // Ou separados por uma vírgula ( , ):
    $values = $message->getHeaderLine('X-Foo');
    // output: string(14) 'value1, value2';
```

No exemplo acima, tanto fazer se você informar `accept`, `ACCEPT` ou mesmo `aCCePt` pois serão transformados em cabeçalhos válidos e trarão o mesmo resultado.

A **PSR-7** estipula que todo cabeçalho deverá retornar uma estrutura no formato:

```
    /* Retorna a seguinte estrutura
        [
            'Header' => [
                'value1'
                'value2'
            ]
        ]
     */
    foreach ($message->getHeaders() as $header => $values)
{ ... }
```

Por especificar uma estrutura de retorno, os consumidores sabem exatamente o que esperar e podem processar os cabeçalhos de uma maneira uniforme - independentemente da implementação.

Mas e quando queremos adicionar um cabeçalho a mensagem - por exemplo, para criar uma requisição para enviar ao cliente?

As mensagems na **PSR-7** são modeladas como values objects; isto significa que qualquer alteração ao estado é essencialmente um valor diferente. Assim, a atribuição de um cabeçalho irá resultar em uma nova instância de mensagem:

```
$new = $message->withHeader('Location', 'http://example.com
');
```

Você pode também, atualizar um valor, adicionar um novo ou remover um outro `header` (cabeçalho):

```
// Atualizando
$message = $message->withHeader('Location', 'http://example
.com');

// Adicionando um novo
$message = $message->withAddedHeader('X-Foo', 'bar');

// Removendo
$message = $message->withoutHeader('X-Foo');
```

O que é a PSR-7

Em tradução livre direta de https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-7-http-message-meta.md.

> *O objetivo da presente proposta consiste em proporcionar um conjunto de interfaces comuns de mensagens HTTP, como descrito na RFC 7230 e RFC 7231, e URIs como descrito na RFC 3986 (no contexto de mensagens HTTP).*

Uma interface serve para informar um conjunto de métodos e atributos que DEVEM ser implementados em uma classe, isso vai garantir que (não importa qual classe) ela tenha o que é necessário para que funcione ao ser usada com outros objetos (classes). Daí o termo

interoperabilidade.

As interfaces

As interfaces aprovadas são 7, cada uma para resolver um problema específico.

1. Psr\Http\Message**MessageInterface**
2. Psr\Http\Message**RequestInterface**
3. Psr\Http\Message**ServerRequestInterface**
4. Psr\Http\Message**ResponseInterface**
5. Psr\Http\Message**StreamInterface**
6. Psr\Http\Message**UriInterface**
7. Psr\Http\Message**UploadedFileInterface**

Dentro de alguns desses problemas, um problema que assombra em PHP é quando precisa trafegar arquivos utilizando o `$_FILES` o php retorna a seguinte estrutura:

```
array(
    'files' => array(
        'name' => array(
            0 => 'file0.txt',
            1 => 'file1.html',
        ),
        'type' => array(
            0 => 'text/plain',
            1 => 'text/html',
```

```
        ),
        /* etc. */
    ),
)
```

Enquanto o esperado, deveria ser algo na seguinte estrutura:

```
array(
    'files' => array(
        0 => array(
            'name' => 'file0.txt',
            'type' => 'text/plain',
            /* etc. */
        ),
        1 => array(
            'name' => 'file1.html',
            'type' => 'text/html',
            /* etc. */
        ),
    ),
)
```

Seguindo o exemplo acima, imaginemos que tenhamos o seguinte trecho HTML:

```
Upload an avatar: <input type="file" name="my-form[details]
[avatars][]" />
```

```
Upload an avatar: <input type="file" name="my-form[details]
[avatars][]" />
```

E a saída é a seguinte:

```
array(
    'my-form' => array(
        'details' => array(
            'avatars' => array(
                'tmp_name' => array(
                    0 => '...',
                    1 => '...',
                    2 => '...',
                ),
                'name' => array(
                    0 => '...',
                    1 => '...',
                    2 => '...',
                ),
                'size' => array(
                    0 => '...',
                    1 => '...',
                    2 => '...',
                ),
                'type' => array(
                    0 => '...',
                    1 => '...',
```

```
                    2 => '...',
                ),
                'error' => array(
                    0 => '...',
                    1 => '...',
                    2 => '...',
                ),
            ),
        ),
    ),
)
```

Seguindo a estrutura da PSR-7 deve retornar um modelo normalizado:

```
array(
    'my-form' => array(
        'details' => array(
            'avatars' => array(
                0 => /* UploadedFileInterface instance */,
                1 => /* UploadedFileInterface instance */,
                2 => /* UploadedFileInterface instance */,
            ),
        ),
    ),
)

// ... Recuperando o objeto
```

```php
$file0 = $request->getUploadedFiles()['files'][0];
$file1 = $request->getUploadedFiles()['files'][1];

printf(
    "Recuperando os arquivos %s e %s",
    $file0->getClientFilename(),
    $file1->getClientFilename()
);

// "Recuperando os arquivos file0.txt e file1.html"
```

Fonte:

- Link Original Baseado
- PHP-FIG / PSR-7
- PSR-7 - O que é isso afinal de contas