Programação Lógica

Programação Web em Prolog

Alexsandro Santos Soares
prof.asoares@gmail.com

25 de setembro de 2020

Bacharelado em Sistemas de Informação
Faculdade de Computação
Universidade Federal de Uberlândia

Sumário

Sumário i

Introdução

Introdução

A partir de agora aprenderemos como tratar os parâmetros recebidos em um pedido.

Isto será útil para, por exemplo, armazenar os dados enviados pelo cliente em um banco de dados.

Nos próximos slides vamos compreender como funciona o protocolo HTTP.

Protocolo HTTP – pedido

Ao digitar um endereço no navegador, daqui em diante chamado de localizador uniforme de recursos ou URL na sigla inglesa, o navegador traduzirá o URL em um pedido com o formato adequado e depois enviará o pedido para o servidor.

Por exemplo, o navegador traduzirá a URL

`http://localhost:8000/exemplo1`

em um pedido similar a este:

```
GET /exemplo1 HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0
Accept: text/html, image/webp
Accept-Language: en-US,en
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Protocolo HTTP – resposta

Ao chegar no servidor o pedido é analisado e, dependendo das circunstâncias, será atendido e a resposta enviada de volta ao navegador.

Por exemplo, para o pedido anterior o servidor poderia retornar a seguinte resposta:

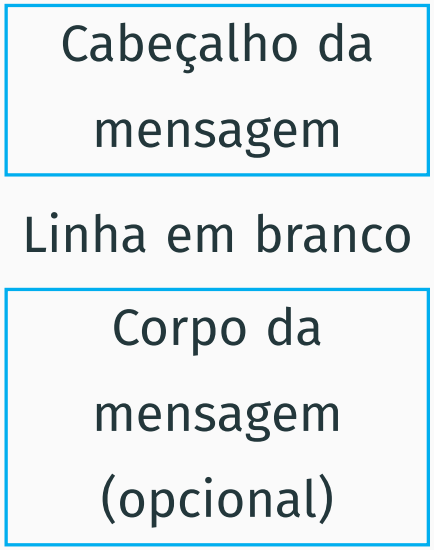
```
HTTP/1.1 200 OK
Date: Fri, 25 Sep 2020 03:11:45 GMT
Content-Type:  text/html; charset=UTF-8
Connection: keep-alive
Content-Length: 790

<!DOCTYPE html>
<html lang="pt-br">
<head>
<title>Exemplo 1</title>
...
```

Mensagens com Pedidos ou Respostas

O cliente e o servidor HTTP comunicam-se enviando mensagens de textos.

De forma geral, uma mensagem HTTP possui o seguinte formato:



Mensagem de pedido HTTP

O formato de uma mensagem HTTP de pedido é o seguinte:

```
Verbo URI HTTP/Versão
Cabeçalho do pedido

Corpo opcional do pedido
```

Onde

- Verbo indica o tipo do pedido sendo enviado: GET, PUT, POST, DELETE, etc.
- URI é a sigla inglesa para o indicador uniforme de recurso que informa a rota do recurso que se deseja acessar no servidor.
- Versão é a versão do protocolo HTTP sendo usada.
- O cabeçalho do pedido é formado por pares chave-valor com informações do pedido, tais como: nome do cliente ou navegador, formatos reconhecidos pelo cliente, tamanho do corpo em bytes, etc.
- Corpo do pedido é o conteúdo da mensagem

Exemplo de mensagem do pedido

```
GET /exemplo1 HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0
Accept: text/html, image/webp
Accept-Language: en-US,en
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

Se o nome possuir múltiplos valores eles são separados por vírgula.

Mensagem de resposta HTTP

O formato de uma mensagem HTTP de resposta é o seguinte:

```
HTTP/Versão Código de resposta Explicação curta
Cabeçalho da resposta

Corpo opcional da resposta
```

Onde

- Versão é a versão do protocolo HTTP sendo usada.
- Código de resposta indica o resultado do servidor para o recurso pedido, por exemplo: 200 indica sucesso, 404 indica recurso não encontrado, etc.
- Explicação curta é uma razão curta para o código de resposta, por exemplo: OK para sucesso, Not Found, Forbidden, etc.
- O cabeçalho da resposta é formado por pares chave-valor com informações da resposta, tais como: data, tipo do conteúdo, tamanho do corpo em bytes, etc.
- Corpo do resposta é o conteúdo da mensagem de resposta.

Exemplo de mensagem do pedido

```
HTTP/1.1 200 OK
Date: Fri, 25 Sep 2020 03:11:45 GMT
Content-Type: text/html; charset=UTF-8
Connection: keep-alive
Content-Length: 790

<!DOCTYPE html>
<html lang="pt-br">
<head>
<title>Exemplo 1</title>}
...
```

Métodos de solicitação

O protocolo HTTP define oito métodos que o cliente pode usar para indicar a ação a ser realizada pelo servidor no recurso especificado. Os principais para nós são

GET usado para obter um recurso do servidor. Ex: o valor de um atributo, as tuplas de uma tabela, o resultado de algum programa, etc.

POST usado para postar dados para o servidor, esperando que ele crie um valor ou um recurso. Ex: criar uma tupla em uma tabela, enviar o conteúdo de um arquivo, etc.

PUT visa a atualização de um recurso no servidor. Ex: atualizar uma tupla, uma parte de um arquivo, etc.

DELETE usado para remover um determinado recurso. Ex: apagar uma tupla em uma tabela, remover algum arquivo, etc.

Pedidos com dados

Se o cliente enviar ao servidor um pedido associado a dados, chamados parâmetros do pedido, ele pode fazê-lo de três formas:

- Anexado à URI como valores de busca.
- No corpo do pedido.
- Como uma combinação dos dois anteriores.

Qualquer que seja a alternativa usada, o pedido também conterá os parametros como variáveis no cabeçalho do pedido.

Enviando parâmetros como valores de busca

Quando o cliente envia os dados como parâmetros de busca, esses são posicionados no final da URI, após um caractere `?`, seguidos de pares `nome=valor` separados pelo `\&`. O formato é

`URI/recurso?string de busca#fragmento`

A string de busca é da forma

$$\text{nome}_1 = \text{valor}_1, \text{nome}_2 = \text{valor}_2, \cdots, \text{nome}_3 = \text{valor}_3$$

Espaços em branco na string de busca são indicados por `%20` ou por `+`. Ex:

`localhost:8000/clientes?id=12&nome=Gal+Costa&prof=cantora`

Normalmente, enviamos parâmetros na string de busca com o verbo GET.

Enviando parâmetros no corpo do pedido

Quando o método de solicitação é o POST os parâmetros são enviados no corpo do pedido e não na string de busca.

Isto é útil quando se deseja enviar grandes quantidades de dados tais como: o conteúdo de um formulário ou um arquivo inteiro.

Exemplo 1 – Post

Vamos apresentar um exemplo na qual o cliente envia um formulário preenchido e o servidor Prolog devolve os dados recebidos na forma de um texto.

```
/* http_read_data está aqui */
:- use_module(library(http/http_client)).

% Liga a rota ao tratador
:- http_handler(root(.), formulário , []).

formulário(_Pedido) :-
   reply_html_page( title('Demonstração de POST'),
    [ form([ action='/receptor', method='POST'],
               [ p([], [ label([for=name],'Nome:'),
                         input([name=name, type=textarea]) ]),
                 p([], [ label([for=email],'Email:'),
                         input([name=email, type=textarea]) ]),
                 p([], input([ name=submit, type=submit, value='Enviar'],
                             []))
        ])]).
```

Exemplo 1 – Post

Note que precisamos incluir a biblioteca `http/http_client`, pois precisaremos do predicado `http_read_data` que lerá os dados do pedido HTTP e os colocará em uma lista.

```
:- http_handler('/receptor', recebe_formulário(Method),
                [ method(Method),
                  methods([post]) ]).

recebe_formulário(post, Pedido) :-
    http_read_data(Pedido, Dados, []),
    format('Content-type: text/html~n~n', []),
    format('<p>', []),
    portray_clause(Dados), % escreve os dados do corpo
    format('</p><p>========~n', []),
    portray_clause(Pedido), % escreve o pedido todo
    format('</p>').
```

O predicado `portray_clause` serve para transformar um termo qualquer em um átomo para depois escrevê-lo na saída padrão.

Exemplo 2 - GET

Vamos agora fazer um exemplo onde os parâmetros são passados na string de busca via GET.

```
/* http_parameters está aqui */
:- use_module(library(http/http_parameters)).
% Liga a rota ao tratador
:- http_handler(root(.), home , []).

home(Pedido) :-
reply_html_page( title('Demonstração de GET'),
           [ \página(Pedido) ]).
```

Exemplo 2 - GET

```
página(Pedido) -->
    { /* É preciso tratar a exceção pois http_parameters gera uma
         uma exceção caso um parâmetro seja inválido */
      catch(
          http_parameters(Pedido,
                          [ % id dever ser um inteiro
                            id(Id,           [integer]),
                            nome(Nome,       [string]),
                            prof(Profissão, [string]),
                           /* Se o parâmetro idade estiver
                              ausente, Idade não ficará ligada     */
                            idade(Idade,    [optional(true),integer]),
                           /* Se o parâmetro nacionalidade estiver
                              ausente, assuma o valor default.     */
                            nacionalidade(Nac, [default(brasileira)])
                          ]),
          _E,
          fail ),
      !,
      % Verifica se a idade foi informada
      (var(Idade) -> Idade = 'desconhecida' ; true)
```

Exemplo 2 - GET

```
    },
    html([ h1('Resposta do servidor para o GET'),
           p('Os parâmetros recebidos foram:'),
           p('Id é ~w' - Id),
           p('Nome é ~w' - Nome),
           p('Profissão é ~w'- Profissão),
           p('Idade é ~w' - Idade),
           p('Nacionalidade é ~w'- Nac)
           ]).

/* Essa página será exibida em caso de erro de validação
   de algum parâmetro */
página(_Pedido) -->
    html([ h1('Erro'),
           p('Algum parâmetro não é válido')
         ]).
```

Testando o exemplo 2 - GET

Podemos testar o exemplo 2 entrando no navegador e digitando:

- `localhost:8000/?id=12&nome=Gal+Costa&prof=cantora`
- `localhost:8000/?id=12&nome=Gal+Costa&prof=cantora&idade=74`
- `localhost:8000/?id=12&nome=Gal+Costa&prof=cantora&idade=Oito`
- `localhost:8000/?id=12&nome=Joss+Stone&prof=cantora&nacionalidade=inglesa`

Consulte a documentação do predicado `http_parameters` para aprender sobre o uso de outras opções de validação:

`https://www.swi-prolog.org/pldoc/doc_for?object=http_parameters/2`

Exemplo 3 – parâmetros sem validação

Este terceiro exemplo mostra uma forma de se obter todos parâmetros enviados, sem realizar validação.

```
:- use_module(library(http/http_parameters)).
home(Pedido) :-
reply_html_page( title('Demonstração de GET'),
           [ \página(Pedido) ]).

página(Pedido) -->
    {   catch(
            http_parameters(Pedido, [], [ form_data(Dados) ]),
            _E,
            fail ),
        !,
        resposta(Dados, HTML)
    },
    html([h1('Resposta do servidor') | HTML]).

página(_Pedido) -->
    html([ h1('Erro'),
           p('Algum parâmetro não é válido') ]).
```

Exemplo 3 – parâmetros sem validação

```
resposta([Nome=Valor | Atribs], [Par|Pares]) :-
    Par = p('~w = ~w'-[Nome,Valor]),
    !,
    resposta(Atribs, Pares).
resposta([],[]).
```

A opção `form_data(Dados)` do predicado `http_parameters/3` retorna todo o conjunto de parâmetros recebidos como uma lista de pares `Nome=Valor`.

Esta opção pode ser usada tanto com um pedido GET quanto por um POST.

JSON

JSON (JavaScript Object Notation - Notação de Objetos JavaScript) é um formato textual, compacto e leve para a troca de dados entre sistemas.

Os tipos de dados básicos do JSON são:

Número um número que pode ter sinal, uma parte fracionária separada por um ponto. Também pode ser usada a notação científica.

String cadeia de caracteres Unicode delimitadas por aspas.

Booleano valores lógicos `true` ou `false`.

Arranjo uma lista de elementos separados por vírgulas. A lista é delimitada por colchetes, como em Prolog.

Objeto uma coleção não ordenada de pares atributo-valor, onde os atributos são strings. Um objeto é delimitado por chaves e os pares são escritos no formato atributo:valor e separados por vírgula.

null usado para indicar um valor vazio ou nulo.

Exemplo de JSON

Abaixo está um exemplo de JSON com uma lista de estudantes e suas notas:

```
{ "Alunos": [

     { "nome": "Aldovandro Cantagalo",
       "notas": [ 20, 15, 35 ]  },

     { "nome": "Anna Karenina",
       "notas": [ 20, 30, 50 ] },

     { "nome": "Pedro de Alvarenga",
       "notas": [ 16, 20, 39 ] }
]}
```

JSON em Prolog

O exemplo do slide anterior é traduzido assim para o Prolog

```
json([Alunos=[json([nome='Aldovandro Cantagalo',
                    notas=[20,15,35]]),
               json([nome='Anna Karenina',
                    notas=[20,30,50]]),
               json([nome='Pedro de Alvarenga',
                    notas=[16,20,39]])]]).
```

O SWI-Prolog possui várias bibliotecas para trabalhar com JSON.

Uma delas é a `http/json` que nos permite ler do formato JSON para um termo Prolog e vice-versa.

Veremos um exemplo de uso no próximo slide

Lendo JSON em Prolog

```
?- use_module(library(http/json)).

?- json_read(current_input, Termo).
|: { "Alunos": [
|:
|:       { "nome": "Aldovandro Cantagalo",
|:         "notas": [ 20, 15, 35 ]  },
|:
|:       { "nome": "Anna Karenina",
|:         "notas": [ 20, 30, 50 ] },
|:
|:       { "nome": "Pedro de Alvarenga",
|:         "notas": [ 16, 20, 39 ] }
|: ]}

Termo = json(['Alunos'=[json([nome='Aldovandro Cantagalo',notas=[20,15,35]]
                      json([nome='Anna Karenina',notas=[20,30,50]]),
                      json([nome='Pedro de Alvarenga',notas=[16,20,39]])]]).
```

Escrevendo para JSON

```
 ?- json_write_dict(current_output,
    json(['Alunos'=[json([nome='Aldovandro Cantagalo',notas=[20,15,35]]),
                    json([nome='Anna Karenina',notas=[20,30,50]]),
                    json([nome='Pedro de Alvarenga',notas=[16,20,39]])]])).
{
  "Alunos": [
    {"nome":"Aldovandro Cantagalo", "notas": [20, 15, 35 ]},
    {"nome":"Anna Karenina", "notas": [20, 30, 50 ]},
    {"nome":"Pedro de Alvarenga", "notas": [16, 20, 39 ]}
  ]
}
```

Para saber mais

Para saber mais

- SWI-Prolog HTTP support library. Disponível em `www.swi-prolog.org/pldoc/doc/_SWI_/library/http/index.html`.

Referências bibliográficas

Referências bibliográficas

- Anne Ogborn. *Creating Web Applications in SWI Prolog*. Disponível em `http://www.pathwayslms.com/swipltuts/html/index.html`