Programação Lógica

Programação Web em Prolog

Alexsandro Santos Soares
prof.asoares@gmail.com

18 de setembro de 2020

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

Sumário

Sumário i

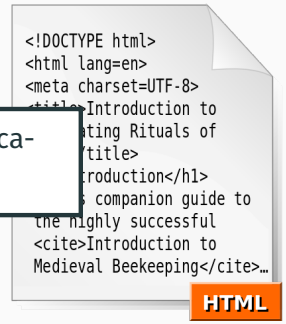
Introdução

Funcionamento de um servidor web

O usuário clica em um link na página

O navegador formata o pedido e o envia para o servidor

O servidor localiza a página



O servidor formata a resposta e a envia para o navegador

O navegador recebe a resposta e a exibe em um formato visual

Criando um servidor web básico

Primeiro exemplo

Para um construir um servidor web em Prolog, digite o seguinte arquivo `ex1.pl`:

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(/, oi, []).

oi(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Oi Mundo!~n').
```

Para executá-lo, entre no Prolog e digite:

```
?- [ex1].

?- servidor(8000).
```

Aponte o seu navegador para `http://localhost:8000/` e veja o resultado.[4]

Detalhamento do primeiro exemplo

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
```

A biblioteca `http/thread_httpd` permite que muitos pedidos sejam tratados concorrentemente. Ela é responsável pela aceitação e gerenciamento de conexões.

Já a biblioteca `http/http_dispatch` permite associar rotas HTTP com os predicados que servirão uma determinada página. Nela estão definidos os predicados `http_dispatch` e `http_handler`.

```
servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).
```

Aqui informamos a porta que o servidor estará escutando.

- Podemos usar qualquer uma que já não esteja em uso.
- Em nossos exemplos usaremos a porta 8000.

Detalhamento do primeiro exemplo

```
:- http_handler(/, oi, []).

oi(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Oi Mundo!~n').
```

`http_handler(Rota,Tratador,Opções)` permite registar qual predicado tratará os pedidos feitos pela rota dada. Por enquanto, não usaremos as opções.

No exemplo, sempre que se acessar o endereço `http:\localhost:8000/` ou simplesmente `http:\localhost:8000`, o servidor passará o pedido para o predicado `oi`, que será o responsável por devolver a resposta para a rota `/`.

De forma geral o endereço de acesso será da forma:

`http:\\localhost:8000\Rota`

e para cada Rota solicitada o servidor deve indicar qual predicado dará a resposta.

Detalhamento do primeiro exemplo

```
oi(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Oi Mundo!~n').
```

Nesta primeira versão a resposta é simples: um texto puro contendo a string Oi Mundo!.

O predicado `oi` recebe o pedido enviado pelo cliente e repassado a ele pelo servidor. Não faremos coisa alguma com o pedido, por enquanto.

Predicados como`oi` que recebem um *pedido* e fornecem uma *resposta* a uma solicitação web são chamados de tratadores (*handlers*).

A resposta ainda não contém HTML e está em muito baixo nível. Logo mudaremos esta situação!

Definindo rotas

Definindo rotas

A medida que um website vai crescendo, também aumentam o número de rotas a serem tratadas.

- Se não formos cuidadosos na estruturação das rotas teremos muitos problemas.

Para ajudar na estruturação de rotas o SWI-Prolog nos fornece os caminhos abstratos.

Para melhor compreendermos o conceito, vamos aumentar nosso exemplo para que as frases também seja escritas em Alemão, Espanhol, Inglês, Francês e Latim. Começaremos com o Alemão no próximo slide.

Segundo exemplo

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), oi, []).
:- http_handler(root(deutsche), hallo, []).

oi(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Oi Mundo!~n').

hallo(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Hallo Welt!~n').
```

Com `http_handler(root(.), oi, [])` informamos ao servidor que o tratador `oi` responderá aos pedidos feitos pela rota raiz (*root*), ou seja, em `http://localhost:8000`, como no primeiro exemplo.

Detalhamento do primeiro exemplo

```
:- http_handler(root(deutsche), hallo, []).

hallo(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Hallo Welt!~n').
```

Agora informamos ao servidor que o tratador `hallo` responderá aos pedidos feitos pelo rota `/deutsche` indicado em Prolog por `root(deutsche)`.

Em nosso exemplo, isso significa que toda vez que acessarmos `localhost:8000/deutsche` o tratador responsável pela resposta será `hallo`.

Note que o predicado `hallo` é essencialmente similar ao predicado `oi`, diferindo apenas na frase em Alemão. Faremos o mesmo para as demais línguas.

Terceiro exemplo

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), oi, []).
:- http_handler(root(deutsche), hallo,   []).
:- http_handler(root(espanol),  hola,    []).
:- http_handler(root(english),  hi,      []).
:- http_handler(root(francais), bonjour, []).
:- http_handler(root(latim),    salve,   []).

oi(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Oi Mundo!~n').

hallo(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Hallo Welt!~n').

hola(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Hola Mundo!~n').

hi(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Hello, world!~n').
```

```
bonjour(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Bonjour le monde!~n').

salve(_Pedido) :-
    format('Content-type: text/plain~n~n'),
    format('Salve orbis terrarum.~n').
```

Para eliminar a repetição de código, modificaremos o terceiro exemplo e criaremos um único tratador para as diversas línguas.

A ideia é receber a rota como uma variável e depois repassar, via unificação, o valor desta variável para o tratador. Isso permitirá que ele decida em qual língua responder.

Quarto exemplo

Quarto exemplo

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(Língua), bem_vindo(Língua) , []).

bem_vindo(Língua, _Pedido) :-
    format('Content-type: text/plain~n~n'),
    cumprimento(Língua, Cumprimento),
    format('~w~n', Cumprimento).

cumprimento('',       'Oi Mundo!'):- !.
cumprimento(deutsche, 'Hallo Welt!'):- !.
cumprimento(espanol,  'Hola Mundo!'):- !.
cumprimento(english,  'Hello, world!'):- !.
cumprimento(francais, 'Bonjour le monde!'):- !.
cumprimento(latim,    'Salve orbis terrarum.'):- !.
cumprimento(Língua,    Desconhecida):-
    format(atom(Desconhecida),
           '~w: é uma língua desconhecida!', [Língua]).
```

Acesse `localhost:8000/latim` para ver a saída. Faça o mesmo com as demais línguas e também inclua uma língua desconhecida, por exemplo, `localhost:8000/italiano`

Definindo novos caminhos abstratos

Além de `root`, já pré definido, o SWI-Prolog nos permite criar novos caminhos abstratos. Primeiro incluímos as diretivas no arquivo:

```
:- multifile http:location/3.
:- dynamic   http:location/3.
```

Isto informa duas coisas ao Prolog:

1. A definição de `http:location/3` estará espalhada em vários arquivos, daí usar a diretiva `multifile`.
2. `http:location/3` é um predicado dinâmico.

Suponha que exista uma rota em seu site para arquivos estáticos e você queira que esta rota seja `/arqs` seguido de `/texto`.

Por exemplo, se o usuário acessar `localhost:8000/arqs/texto` ele será direcionado pelo servidor para o predicado `arquivo_estático` que, de fato, enviará o arquivo para o usuário.

Quinto exemplo

Abaixo está a parte do arquivo que implementa esta funcionalidade

```
:- multifile http:location/3.
:- dynamic   http:location/3.

http:location(arquivos, '/arqs', []).

:- http_handler(arquivos(texto), arquivo_estático , []).

arquivo_estático(_Pedido):-
    format('Content-type: text/plain~n~n'),
    format('Alguma dia eu enviarei o arquivo~n').
```

Se o usuário acessar `http://localhost:8000/arqs/texto` ele será direcionado para `arquivo_estático`.

A vantagem de usar caminhos abstratos é que se no futuro movermos os arquivos para algum outro lugar no website basta atualizar um único lugar:

```
http:location(arquivos, '/meu_novo_caminho', []).
```

Gerando HTML

Sites estáticos e sites dinâmicos

Até agora só servimos arquivos textos puros. Aprenderemos agora como servir HTML.

Podemos classificar os websites quanto a forma de fornecer os conteúdos de suas páginas em dois tipos:

estático que vem com um número fixo de páginas, cada uma possuindo um formato específico. Esse formato é estático e não muda em resposta às ações do usuário. As páginas normalmente são criadas apenas com HTML e CSS.

dinâmico pode exibir diferentes conteúdos e interagir com o usuário. As páginas usam além de HTML e CSS, também JavaScript no lado do cliente e linguagens de script no lado servidor.

A classificação acima é muito estrita e, em geral, os websites usam uma mistura das duas formas de geração de conteúdo.

Páginas dinâmicas

Existem duas escolas para a geração dinâmica de HTML:

- A escola dos gabaritos (*templates*) que permite editar código HTML comum, embutindo trechos de código entre delimitadores, como por exemplo `<% ... %>`. Este é o caso de PhP, JSP, ASP, etc.
  - A ideia é colocar código de programação dentro do HTML.
- A escola da geração dinâmica de páginas web que possui uma representação própria do HTML e permite a mistura de código e HTML.
  - A ideia é colocar HTML dentro do código de programação.

Em SWI-Prolog podemos trabalhar com as duas opções:

- Para os gabaritos temos o Simple-Template e o Prolog Web Pages (PWP).
- Na geração dinâmica temos o `html//1` que já vem com a distribuição do SWI-Prolog.

Vamos trabalhar com o `html//1` por agora.

Gerando HTML diretamente

Podemos servir HTML apenas imprimindo (`ex6.pl`):

```
:- use_module(library(http/http_dispatch)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), bem_vindo , []).

bem_vindo(_Pedido) :-
    format('Content-type: text/html~n~n'),
    format('<html><head><title>Bem-vindo</title></head><body><h2>Uma página
    simples</h2><p>com algum texto.</p></body></html>~n').
```

Duas coisas a se notar:

- o tipo do conteúdo enviado ao cliente foi modificado para `text/html`
- o conteúdo enviado é uma página HTML completa.

Usando print_html

Existe uma forma um pouco mais interessante de gerar HTML com `print_html`, que recebe uma lista de trechos HTML e os converte em uma string contendo HTML.

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/html_write)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), bem_vindo , []).

bem_vindo(_Pedido) :-
    format('Content-type: text/html~n~n'),
    print_html(['<html>',
                '<head>',
                '<title>', 'Bem-vindo', '</title>',
                '</head>',
                '<body>',
                '<h2>', 'Uma página web simples', '</h2>',
                '<p>', 'com algum texto.', '</p>',
                '</body>',
                '</html>']).
```

Note a inclusão da biblioteca `http/html_write` que permite o uso do predicado `print_html/1`.

Representando HTML em SWI-Prolog

O SWI-Prolog possui três representações para o HTML:

HTML atomizado é um átomo ou string com HTML nele. Ex:

```
'<p>Algum texto</p>'
```

HTML tokenizado o HTML é representado como uma lista de átomos. ex:

```
['<p>', 'Algum texto.', '</p>'].
```

HTML termificado o HTML é representado como um termo. Ex:

```
p([], 'Algum texto').
```

A nossa principal ferramenta para gerar HTML é `html//1` que é uma DCG que recebe como argumento um HTML termificado e gera um HTML tokenizado. Por exemplo, a página HTML do último exemplo poderia ser gerada assim:

```
phrase( html([ head(title('Bem-vindo')),
               body([ h1('Uma página web simples'),
                      p('com algum texto.') ])
             ]),
             HtmlTokenizado,
             []).
```

HTML termificado

```
phrase( html([ head(title('Bem-vindo')),
               body([ h1('Uma página web simples'),
                      p('com algum texto.') ])
             ]),
             HtmlTokenizado,
             []).
```

O predicado `phrase` é pré-definido e recebe como primeiro argumento uma DCG, no caso, o predicado `html//1` que é uma linguagem de domínio específico (DSL) para definir HTML.

O segundo argumento é o HTML tokenizado referente ao primeiro argumento:

```
['<html>',
     '<head>', '<title>', 'Bem-vindo', '</title>', '</head>',
     '<body>',
          '<h2>', 'Uma página web simples', '</h2>',
          '<p>',  'com algum texto.', '</p>',
     '</body>',
 '</html>']
```

Oitavo exemplo

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/html_write)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), bem_vindo , []).

bem_vindo(_Pedido) :-
    phrase( html([ head(title('Bem-vindo')),
                   body([ h1('Uma página web simples'),
                          p('com algum texto.') ])
                 ]),
            HtmlTokenizado,
            []),
    format('Content-type: text/html~n~n'),
    print_html(HtmlTokenizado).
```

Nono exemplo

Para mostrar que o primeiro argumento de `phrase` é uma DCG de verdade, vamos separar a geração da página em um predicado próprio.

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/html_write)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), bem_vindo , []).

bem_vindo(_Pedido) :-
    phrase( página,
            HtmlTokenizado,
            []),
    format('Content-type: text/html~n~n'),
    print_html(HtmlTokenizado).

página -->
    html([html([ head(title('Bem-vindo')),
                 body([ h1('Uma página web simples'),
                        p('com algum texto.') ])
               ])]).
```

Na DCG `página`, o predicado `html//1` mais externo é o predicado que recebe o HTML termificado e o transforma em um HTML tokenizado.

O `html` mais interno é o tag de abertura de um documento HTML.

O predicado reply_html_page

Podemos simplificar ainda mais a geração da página usando o predicado `reply_html_page` que cuida de todo o processamento do HTML termificado e gera uma resposta completa para o servidor, incluindo o cabeçalho `Content-type`.

Com ele, a página anterior é gerada assim:

```
:- use_module(library(http/thread_httpd)).
:- use_module(library(http/http_dispatch)).
:- use_module(library(http/html_write)).

servidor(Porta) :-
    http_server(http_dispatch, [port(Porta)]).

:- http_handler(root(.), bem_vindo , []).

bem_vindo(_Pedido) :-
    reply_html_page(
        [title('Bem-vindo')],
        [ h1('Uma página web simples'),
          p('com algum texto.') ]).
```

Para saber mais

Para saber mais

- Thiago Verney. *Sites estáticos e sites dinâmicos: quais as diferenças?*. Disponível em `https://tudosobrehospedagemdesites.com.br/sites-estaticos-e-sites-dinamicos/`

Referências bibliográficas

Referências bibliográficas

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