Programação Lógica

Programação Web em Prolog

Alexsandro Santos Soares
prof.asoares@gmail.com

23 de setembro de 2020

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

Sumário

Sumário i

Sintaxe do HTML termificado

Sintaxe do HTML termificado

O HTML usa um termos com aridades 1 ou 2 para cada *tag* HTML.

O argumento de termos com aridade 1 é considerado um conteúdo (*innerHTML*) do respectivo tag. Exemplos:

HTML termificado	HTML puro
`h1('Uma página web simples')`	`<h1>Uma página web simples</h1>`
`p(b('Texto negritado'))`	`<p><b>Texto negritado</b></p>`
`div(p(b('Negrito')))`	`<div> <p><b>Negrito</b></p> </div>`

Se o argumento for uma lista, os itens são convertidos individualmente e depois concatenados:

HTML termificado	HTML puro
`div([p('Um'), p('Dois')])`	`<div> <p>Um</p> <p>Dois</p> </div>`
`p['palavra 1', 'palavra 2']`	`<p>palavra 1 palavra 2</p>`

Sintaxe do HTML termificado

O encaixamento de texto em uma string é feito no estilo do `format/2`:

```
p('Na tradução de romano para ~w: I é ~d, II é ~d.'-[decimal, 1, 2])
```

é traduzido para

```
<p>Na tradução de romano para decimal: I é 1 e II é 2.</p>
```

Se o HTML termificado possuir aridade 2:

- no primeiro argumento estão os atributos.
- no segundo, o conteúdo (*innerHTML*).

Exemplos:

termo: `p(class=classe, 'Texto')`

HTML: `<p class="classe">Texto</p>`

termo : `p([class=classe, style='font-size: 15px'], 'Texto')`

HTML: `<p class="classe" style="font-size: 15px">Texto</p>`

Atributos

Os atributos podem ser especificados tanto por pares `Atributo=Valor`, como em `class=cliente` quanto por termos `Atributo(Valor)`, como em `class(cliente)`.

A última forma é preferível para evitar problemas com as prioridades dos operadores.

Assim, qualquer um dos termos abaixo

```
img([src='tom.png', class=gato, height=128, width=128, alt='Tom'], [])
img([src('tom.png'), class(gato), height(128), width(128), alt('Tom')],[])
```

produzirá o mesmo HTML:

```
<img src="tom.png" class="gato" height="128" width="128" alt="Tom">
```

Atributos

Um atributo com múltiplos valores pode ser expresso de várias formas:

```
button([ class('btn btn-primary'),         type(submit)], 'Enviar')

button([ class=['btn', 'btn-primary'],     type(submit)], 'Enviar')

button([ class(['btn', 'btn-primary']),    type(submit)], 'Enviar')

button([ class('btn' + ' ' + 'btn-primary'),  type(submit)], 'Enviar')
```

Todos os termos acima produzem o mesmo HTML:

```
<button class="btn btn-primary" type="submit">Enviar</button>
```

No próximo slide veremos um exemplo completo colocando tudo isto junto.

Exemplo 1

```
:- 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(.), home , []).

home(_Pedido) :-
    reply_html_page(
        [title('Exemplos')],
        [ div(class(container),
              [ h1('Exemplos com SWI-Prolog'),
                h2(class(['my-5', 'text-center'], 'Exemplo 1')),
                div(class(row),
                    [ div(class('col-md-8'), 'Conteúdo Principal'),
                      div(class('col-md-4'), 'Barra Lateral')])
              ])]).
```

Agora que nossos exemplos estão ficando maiores, omitirei nos próximos slides o trecho que sempre se repete em cada arquivo.

Por exemplo, omitirei o trecho abaixo:

```
:- 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(.), home , []).
```

Entretanto, os códigos estarão completos nos arquivos da aula.

Inclusão

Inclusão

Inclusão é um mecanismo do SWI-Prolog que permite o encapsulamento de código de geração de HTML.

Com isso podemos criar componentes reutilizáveis de páginas web.

A inclusão é marcada por uma \. Se o argumento de \ for um termo, ele será tratado como uma DCG e expandido para HTML tokenizado.

Por exemplo, a linha

```
\trecho_de_codigo_html
```

Chamará a DCG

```
trecho_de_codigo_html  -->
   html([p('Algum texto)]).
```

Inclusão com argumentos

Como estamos chamando um predicado de uma DCG, podemos incluir argumentos também. Por exemplo

```
\trecho_com_argumento('Este texto em negrito)
```

chamará a DCG

```
trecho_com_argumento(Texto) -->
   html([p(['Mais um trecho: ', b(Texto)])]).
```

Podemos pensar no operador de inclusão `\` como uma frase que diz *entrando no mundo do HTML tokenizado*.

Colocaremos isso em um exemplo no próximo slide.

Exemplo 2

```
home(_Pedido) :-
    reply_html_page(
        [ title('Exemplos')],
        [ \página ]).

página -->
    html([ div(class(container),
                [ h1('Exemplos de inclusão com SWI-Prolog'),
                  \exemplo1,
                  \exemplo2
                ])
         ]).


exemplo1 -->
    html([ h2(class("my-5 text-center"),'Exemplo 1'),
           div(class("container my-5"),
               div(class(row),
                   [ div(class('col-md-8'), 'Conteúdo Principal'),
                     div(class('col-md-4'), 'Barra Lateral')]))]).
```

Continuação do exemplo

```
exemplo2 -->
    html([h2(class("my-5 text-center"),'Exemplo 2'),
          div(class("container"),
              div(class(row),
                  [div(class('col-md-8 text-center'),
                       ['Cabeçalho',
                        div(class="row text-center",
                            [div(class="col-md-6",
                                 ['Artigo',
                                  p('Texto do artigo 1.')]),
                             div(class="col-md-6",
                                 ['Artigo',
                                  p('Texto do artigo 2.')]),
                             span('Rodapé')])]),
                   div(class('col-md-4'), 'Barra Lateral')]))]).
```

Note o uso de `html//1` que recebe HTML termificado e devolve HTML tokenizado.

Estilização

Estilização

Todos os exemplos vistos até agora não possuem nenhuma forma de estilização fornecida pelo CSS.

Desde que a estilização de um website normalmente é padronizada para todas as páginas, temos que encontrar uma forma de evitar repetir o código da estilização em toda e qualquer página.

Até agora usamos o predicado `reply_html_page` com aridade 2. Entretanto, existe uma variante com aridade 3, feita exatamente para resolver problemas como o do parágrafo anterior. Em `reply_html_page(Estilo, Cabeça,Corpo)`:

Estilo é o estilo que deve ser aplicado à pagina gerada.

Cabeça é o conteúdo a ser incluído na cabeça do HTML (*head*).

Corpo é o conteúdo a ser incluído no corpo do HTML (*body*).

Exemplo 3

Para exemplificar, assuma que em todas as páginas se deseje que a frase *O Site da Página Web Simples* fique no topo. Poderemos proceder assim:

Primeiro podemos definir um estilo para o topo da página:

```
:- multifile
       user:body//2.

% Corpo será incluído
user:body(estilo_topo, Corpo) -->
        html(body([ div(id(topo), h1('O Site da Página Web Simples')),
                    div(id(conteudo), Corpo)
                  ])).
```

Depois, podemos incluir o estilo na página:

```
home(_Pedido) :-
    reply_html_page(
        estilo_topo,   % define o estilo da página
        [ title('Exemplos')],
        [ \página ]).
```

Continuação do exemplo 3

```
user:body(estilo_topo, Corpo) -->
        html(body([ div(id(topo), h1('O Site da Página Web Simples')),
                    div(id(conteudo), Corpo)
                  ])).
```

```
home(_Pedido) :-
    reply_html_page(
        estilo_topo,   % define o estilo da página
        [ title('Exemplos')],
        [ \página ]).

página -->
    html([ h2('Cabeçalho específico da página'),
           p('com estilização')]).
```

O estilo `estilo_topo`, definido com `user:body`, sempre receberá o corpo da página como segundo argumento.

Para o exemplo em questão, o HTML tokenizado do predicado `página` será embutido na *div* com id=conteudo.

Página HTML gerada para o exemplo 3

```
<!DOCTYPE html>
<html>
  <head>
    <title>Exemplos</title>
    <meta http-equiv="content-type" content="text/html; charset=UTF-8">
  </head>
  <body>
    <div id="topo">
      <h1>O Site da Página Web Simples</h1>
    </div>
    <div id="conteudo">

      <h2>Cabeçalho específico da página</h2>

      <p>com estilização</p>
    </div>
  </body>
</html>
```

Mailman

Mailman (carteiro) é uma funcionalidade que permite criar HTML em um ponto de um módulo para depois utilizá-lo em algum outro lugar.

Ele será coberto aqui, pois ele é frequentemente utilizado para colocar conteúdo no *head* da página (css, javascript, etc.).

- Ele possui muitos outros usos, como veremos.

Para entender o conceito do *mailman* imagine o seguinte cenário: há uma página como uma área de navegação no topo que é gerada dinamicamente. Para organizar o código, colocaremos esta informação em uma barra de navegação via uma inclusão.

A primeira versão da página é mostrada no próximo slide.

Exemplo 4 - primeira versão

```
home(_Pedido) :-
    reply_html_page(
        [ title('Exemplos')],
        [ \página ]).

página -->
  html([ h1('Demonstração do Mailman'),
         div(\menu_de_navegação),
         p('O corpo vai aqui')
       ]).

menu_de_navegação -->
  {
     findall(Nome, item(Nome, _), Nomes),
     maplist(item_topo, Nomes, BotõesTopo)
  },
  html(BotõesTopo).

item_topo(Nome,
          a([href=Link, class=topo], Nome)):-
   item(Nome, Link).
```

```
item('Início', '/home').
item('Sobre',  '/sobre').
item('Vendas', '/vendas').
```

HTML gerado para o exemplo 4

Podemos verificar o HTML gerado de duas formas:

1. Apontando o navegador para `localhost:8000` e depois verificar o HTML.
2. Chamando o tratador dentro do SWI-Prolog e observar o HTML impresso.

Para o nosso exemplo, faremos assim:

```
?- [ex4].

?- home(_).
Content-type: text/html; charset=UTF-8

<!DOCTYPE html>
<html>
<head>
   <title>Exemplos</title>

   <meta http-equiv="content-type" content="text/html; charset=UTF-8">
</head>
```

HTML gerado para o exemplo 4 – continuação

```
<body>

   <h1>Demonstração do Mailman</h1>

   <div>
      <a href="/home" class="topo">Início</a>
      <a href="/sobre" class="topo">Sobre</a>
      <a href="/vendas" class="topo">Vendas</a>
   </div>

   <p>O corpo vai aqui</p>
</body>
</html>
```

Continuando o exemplo, assuma agora que o conteúdo da página ficou muito comprido e para facilitar a navegação se deseje colocar o menu também na base da página, mas com uma estilização diferente.

Já temos a lista de itens de navegação necessários e queremos evitar a duplicação de trabalho para fazer a lista de itens para a base.

Exemplo 5

Como vimos, o *Mailman* nos possibilita gerar HTML tokenizado em um lugar e postá-lo para outro.

Para postar usamos `\html_post` e para receber usamos `\html_receive`.

Para o exemplo, enviaremos os itens de navegação da base com

```
\html_post(nav_base, BotõesBase)
```

E os receberemos em uma `div` com

```
div(\html_receive(nav_base))
```

Observe que o primeiro argumento tanto de `\html_post` quanto de `\html_receive` é o nome da caixa postal para onde o código será postado ou lido, respectivamente.

Todo o trecho modificado para o exemplo é mostrado no próximo slide.

Exemplo 5

```
página -->
    html([ h1('Demonstração do Mailman'),
           div(\menu_de_navegação),
           p('O corpo vai aqui'),
           div(\html_receive(nav_base)) /* recebe */
         ]).

menu_de_navegação -->
    {
      findall(Nome, item(Nome, _), NomesBotões),
      maplist(item_topo, NomesBotões, BotõesTopo),
      maplist(item_base, NomesBotões, BotõesBase)
    },
    html([\html_post(nav_base, BotõesBase) | BotõesTopo]). /* posta */

item_topo(Nome, a([href=Link, class=topo], Nome)) :-
    item(Nome, Link).

/* Note que a classe é diferente: base */
item_base(Nome, a([href=Link, class=base], Nome)) :-
    item(Nome, Link).
```

Exemplo 5 – corpo do HTML gerado

```
<h1>Demonstração do Mailman</h1>

<div>
   <a href="/home"   class="topo">Início</a>
   <a href="/sobre"  class="topo">Sobre</a>
   <a href="/vendas" class="topo">Vendas</a>
</div>

<p>O corpo vai aqui</p>

<div>
   <a href="/home"   class="base">Início</a>
   <a href="/sobre"  class="base">Sobre</a>
   <a href="/vendas" class="base">Vendas</a>
</div>
```

Razões para se usar o mailman

Os usos mais comuns do *mailman* são:

- um elemento da página necessita que algo esteja na cabeça do HTML (*head*) tais como: uma biblioteca CSS, um trecho da JavaScript. De fato, o nome de caixa postal head já vem pré definido.
- um elemento, digamos uma lista de postagens de blogs, cujo conteúdoé acumulado durante a geração de uma página e que necessite que o *link* para cada item acumulado apareça próximo ao topo da página.

Introduzido CSS e JavaScript

Agora que aprendemos sobre o mailman vamos utilizá-lo para incluir CSS externo em nossas páginas.

Usarei como exemplo o Bootstrap 5, mas poderia ser qualquer outra biblioteca CSS ou arquivo CSS.

Para os exemplos que se seguirão, imagine que o diretório esteja organizado da seguinte forma:

```
.
├── css
│   ├── bootstrap.min.css
│   └── estilo.css
├── js
│   └── bootstrap.bundle.min.js
└── ex6.pl
```

No próximo slide é mostrada uma página HTML básica usando o Bootstrap 5.

Página web básica com Bootstrap 5

```
<!doctype html>
<html lang="pt-br">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <link rel="stylesheet" href="css/bootstrap.min.css">
    <link rel="stylesheet" href="css/estilo.css">
    <title>Bootstrap 5 básico</title>
  </head>
  <body>
    <h1>Bootstrap 5</h1>

    <script src="js/bootstrap.bundle.min.js"></script>
  </body>
</html>
```

Exemplo 6 - parte I

```
/* html_requires está aqui                              */
:- use_module(library(http/html_head)).

/* serve_files_in_directory está aqui                   */
:- use_module(library(http/http_server_files)).

/* Localização dos diretórios no sistema de arquivos */
:- multifile user:file_search_path/2.
user:file_search_path(dir_css, 'css').
user:file_search_path(dir_js,  'js').
```

Precisamos de duas novas bibliotecas:

- http/html_head define `html_requires` que é usado para incluir um recurso ou lista de recursos, tais como arquivos CSS ou JavaScript, na cabeça do HTML.
- http/http_server_files define `serve_files_in_directory` que permite servir arquivos estáticos de um determinado diretório.

O predicado file_search_path associa um apelido a um diretório ou subdiretório no sistema de arquivos.

Exemplo 6 - parte II

```
/* Liga as rotas aos respectivos diretórios          */
:- http_handler(css(.),
                serve_files_in_directory(dir_css), [prefix]).
:- http_handler(js(.),
                serve_files_in_directory(dir_js), [prefix]).
```

Aqui definimos as rotas e os tratadores para páginas estáticas.

As rotas css e js já são pré definidas, só precismos indicar os tratadores. Assim, sempre que ao servidor for solicitado um arquivo da rota:

- `/css`, o arquivo será buscado em `dir_css`, que é o subdiretório `./css`.
- `/js`, o arquivo será buscado em `dir_js`

A opção prefix indica que a rota dada é apenas um prefixo para a rota total. Ex:

- na rota `/css/estilo.css`, o prefixo é `/css` e assim o servidor procurará o arquivo `estilo.css` no subdiretório `dir_css` e o enviará para o cliente.

Exemplo 6 - parte III

```
/* Liga a rota ao tratador                                  */
:- http_handler(root(.), home , []).

home(_Pedido) :-
    reply_html_page(
        [ title('Bootstrap 5 básico')],
        [ \página ]).
```

O tratador para a página dinâmica acessada pela rota raiz é `home`.

O corpo da página dinâmica será criado pela DCG `página`, definida no próximo slide.

Exemplo 6 - parte IV

```
página -->
    html([ \html_post(head,
                 [ meta([name(viewport),
                   content('width=device-width, initial-scale=1')])]),
           \html_root_attribute(lang,'pt-br'),
           \html_requires(css('estilo.css')),
           \html_requires(css('bootstrap.min.css')),

           h1('Bootstrap 5'),

           script([ src('js/bootstrap.bundle.min.js'),
                    type('text/javascript')], [])
         ]).
```

html_post(head,...) serve para enviar conteúdo para a cabeça do HTML.

html_root_attribute serve para definir atributos do *tag* html. No exemplo, será gerado o seguinte HTML:

```
<html lang="pt-br">
```

Exemplo 6 - parte IV

```
página -->
    html([ \html_post(head,
                 [ meta([name(viewport),
                   content('width=device-width, initial-scale=1')])]),
           \html_root_attribute(lang,'pt-br'),
           \html_requires(css('estilo.css')),
           \html_requires(css('bootstrap.min.css')),

           h1('Bootstrap 5'),

           script([ src('js/bootstrap.bundle.min.js'),
                    type('text/javascript')], [])
         ]).
```

As duas invocações de html_requires servem para incluir na cabeça do HTML os dois arquivos .css que usaremos. Note que rota usada para eles é css.

O último elemento serve para incluir um arquivo JavaScript no final do corpo do HTML.

Exemplo 7 – Criando um gabarito

Qualquer página que use o Bootstrap terá, no mínimo, o formato visto no slide anterior, mudando apenas o interior do corpo do HTML.

Para evitar repetir este código e padronizar as próximas páginas, criaremos um gabarito de nome `bootstrap`.

```
:- multifile  user:body//2.

user:body(bootstrap, Corpo) -->
       html(body([ \html_post(head,
                     [ meta([name(viewport),
                       content('width=device-width, initial-scale=1')])]),
                   \html_root_attribute(lang,'pt-br'),
                   \html_requires(css('bootstrap.min.css')),

                   Corpo,

                   script([ src('js/bootstrap.bundle.min.js'),
                            type('text/javascript')], [])
                 ])).
```

Exemplo 7 – Site multi-páginas

Vamos reutilizar o gabarito anterior para criar um site com 4 páginas: a página inicial e mais três outras com exemplos de uso do Bootstrap.

Primeiro vamos definir os tratadores para as quatro rotas:

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

:- http_handler(root(exemplo1), exemplo1 , []).
:- http_handler(root(exemplo2), exemplo2 , []).
:- http_handler(root(exemplo3), exemplo3 , []).
```

Exemplo 7 – Página de entrada

```
home(_Pedido) :-
    reply_html_page(
        bootstrap,
        [ title('Bootstrap 5 básico')],
        [ div(class(container),
              [ h1('Exemplos usando Bootstrap 5 e SWI-Prolog'),
                nav(class(['nav', 'flex-column']),
                    [ \link_exemplo(1),
                      \link_exemplo(2),
                      \link_exemplo(3) ])
              ])
        ]).


link_exemplo(N) -->
    html(a([ class(['nav-link']),
             href('/exemplo~d' - N)],
           'Exemplo ~d' - N)).
```

Exemplo 7 – Página do exemplo 1

```
exemplo1(_Pedido) :-
    reply_html_page(
        bootstrap,
        [ title('Exemplo 1')],
        [ div(class(container),
              [ \html_requires(css('estilo.css')),
                h2(class("my-5 text-center"),
                   'Exemplo 1 com Bootstrap 5 e SWI-Prolog'),
                div(class("container my-5"),
                    div(class(row),
                        [div(class('col-md-8'),
                             'Conteúdo Principal'),
                         div(class('col-md-4'),
                             'Barra Lateral')])),
                \retorna_home  ])]).

retorna_home -->
    html(div(class(row),
             a([ class(['btn', 'btn-primary']), href('/')],
                'Voltar para o início'))).
```

Exemplo 7 – Página do exemplo 2

```
exemplo2(_Pedido) :-
    reply_html_page(
        bootstrap,
        [ title('Exemplo 2')],
        [ div(class(container),
              [ \html_requires(css('estilo.css')),
                h2(class("my-5 text-center"),
                   'Exemplo 2 com Bootstrap 5 e SWI-Prolog'),
                div(class('container my-5'),
                    div(class(row),
                        [div(class('col-md-8 text-center'),
                             ['Cabeçalho',
                              div(class="row text-center",
                                  [div(class="col-md-6",
                                       [ 'Artigo 1',  p('Parágrafo 1.')]),
                                   div(class="col-md-6",
                                       [ 'Artigo 2',  p('Parágrafo 2.')]),
                                   span('Rodapé')])]),
                         div(class('col-md-4'),
                             'Barra Lateral')])),
                \retorna_home ])]).
```

Exemplo 7 – Página do exemplo 3

```
exemplo3(_Pedido) :-
    reply_html_page(
        bootstrap,
        [ title('Exemplo 3')],
        [ div(class(container),
              [ \html_requires(css('estilo.css')),
                h2(class("my-5 text-center"),
                   'Exemplo 3 com Bootstrap 5 e SWI-Prolog'),
                div(class('container my-5'),
                    div(class(row),
                        \colunas(12))),
                \retorna_home
              ])]).
```

Exemplo 7 – Página do exemplo 3

```
exemplo3(_Pedido) :-
    reply_html_page(
        bootstrap,
        [ title('Exemplo 3')],
        [ div(class(container),
              [ \html_requires(css('estilo.css')),
                h2(class("my-5 text-center"),
                   'Exemplo 3 com Bootstrap 5 e SWI-Prolog'),
                div(class('container my-5'),
                    div(class(row),
                        \colunas(12))),
                \retorna_home
              ])]).
```

Na página deste exemplo queremos 12 linhas praticamente iguais:

- A diferença entre uma e outro é o texto interno: *Coluna 1*, *Coluna 2*, . . . , *Coluna 12*.

Exemplo 7 – Página do exemplo 3

Ao invés de copiar e colar o elemento HTML 12 vezes e realizar a substituição do texto, escreveremos um predicado `colunas(N)` que recebe o número `N` de colunas a serem geradas e devolve uma lista com os `N` elementos HTML já prontos.

```
colunas(Max) -->
    {   findall(div(class(['col-xxl-1', 'col-xl-2', 'col-lg-3',
                           'col-md-4',  'col-sm-6', 'col-xs-12']),
                    ['Coluna ', I]),
                between(1,Max,I),
                Colunas)
    },
    html(Colunas).
```

O predicado `between(Início,Fim,Num)` vai gerado os números *Início*, *Início+1*, *Início+2* até *Fim*, um por vez, unificando o número gerado com `Num`. Por exemplo:

```
?- between(1, 3, I).
I = 1 ;
I = 2 ;
I = 3.
```

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`