EPI: Interfejs Graficzny 2008/2009
Żądania HTTP oraz obsługa stanu

Agnieszka Figiel, Aleksander Pohl

6 stycznia 2009

GET oraz POST

- różnica techniczna:
  - treść żądania GET jest zakodowane w URL-u
  - treść żądania POST jest wysyłana w ciele wiadomości
- rekomendowane użycie wg HTTP 1.1
  - GET tylko do pobierania informacji; bezpieczne, idempotentne
  - POST zarówno do pobierania informacji, jak i akcji posiadających efekty uboczne (modyfikacja danych, usuwanie, etc.)
- http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

Wysyłanie formularzy – GET

```
<form method="get">
  Title:<input name="title_get">
  <input type="submit">
</form>
```

```
GET /form_test.html?title_get=yada HTTP/1.1
```

Wysyłanie formularzy – POST

```
<form method="post">
  Title:<input name="title_post">
  <input type="submit">
</form>
```

```
POST /form_test.html HTTP/1.1
...
Content-Type: application/x-www-form-urlencoded
Content-Length: 15
title_post=yada
```

Linki – GET

```
<a href="link_test.html?title=yada">link test</a>
```

```
GET /link_test.html?title=yada HTTP/1.1
```

Linki – POST (mniej więcej)

```
<a href="link_test.html?title=yada" onclick="
  var f = document.createElement('form');
  this.parentNode.appendChild(f);
  f.method = 'POST';
  f.action = this.href;
  f.submit();
  return false;
">link test</a>
```

```
POST /link_test.html?title=yada HTTP/1.1
```

GET i POST w formularzach – RoR

- POST:

```
<% form_tag :action => 'create' do %>
  ...
<% end %>
```

- GET:

```
<% form_tag({:action => 'show'}, :method=>:get) do %>
  ...
<% end %>
```

GET i POST w linkach – RoR

- GET

```
<%= link_to 'Back', :action => 'list' %>
```

- POST

```
<%= link_to 'Back', {:action => 'list'}, :method => :post %>
```

Formularz – przykład

```
<% form_for :book do |f|%>
  <%= f.text_field :title %>
  <%= f.select :author_id, @authors %>
  <%= f.submit "Create" %>
<% end %>
```

```
<form action="/books/create" method="post">
  <input name="book[title]" size="30" type="text"/>
  <select name="book[author_id]">
    <option value="1">Alan Alexander Milne</option>
    <option value="2">J.K. Rowling</option>
  </select>
  <input name="commit" type="submit" value="Create"/>
</form>
```

Żądanie

```
POST /books/create HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; pl; rv:1.8.1.9)
  Gecko/20071025 Firefox/2.0.0.9
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,
  text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: pl,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-2,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://localhost:3000/books/new
Cookie: _library_session_id=d72a8d72e181f333bf4f5d01d29e163d
Content-Type: application/x-www-form-urlencoded
Content-Length: 67
book%5Btitle%5D=Winnie+the+Pooh&book%5Bauthor_id%5D=1&commit=Create
```

Żądania i stan w RoR – kontroler

specjalne metody kontrolera:

- request
- headers
- params
- response
- template
- session
- cookies
- flash

Żądanie

- request.host, request.port, ...
- request.method, request.get?, request.post?, ...
- request.env – hasz zawierający zmienne środowiskowe

```
<% for key, value in request.env %>
  <%= key %></td><td><%= value %></td></tr>
<% end %>
```

Żądania GET i POST

- akcja, która nie jest bezpieczna (destroy, update, create) powinna weryfikować, czy żądanie zostało wysłane w sposób poprawny (za pomocą POST, a nie GET)
- pierwszy sposób (w kontrolerze):

```
verify :method => :post, :only => [ :destroy, :create, :update ],
       :redirect_to => { :action => :index }
```

- drugi sposób (w akcji):

```
def some_action
  if !request.post?
    redirect_to :action => :list and return
  end
  ...
end
```

Parametry żądania (params)

```
<% form_for :book do |f|%>
  <%= f.text_field :title %>
  <%= f.select :author_id, @authors %>
  <%= f.submit "Create" %>
<% end %>
```

```
<form action="/books/create" method="post">
  <input name="book[title]" size="30" type="text"/>
  <select name="book[author_id]">
    <option value="1">Alan Alexander Milne</option>
    <option value="2">J.K. Rowling</option>
  </select>
  <input name="commit" type="submit" value="Create"/>
</form>
```

```
{
  "commit" => "Create", "action"=>"create", "controller"=>"books",
  "book"=> {
    "title"=>"Harry Potter and the Philosopher's Stone", "author_id"=>"2"
  }
}
params["book"]["title"]
```

Stan

- protokół HTTP jest bezstanowy
- potrzebny jest dodatkowy mechanizm do obsługi stanu
- np. do przechowywania informacji o zalogowany użytkowniku, zawartości jego koszyka, itp.
- obsługa sesji
  - zagnieżdżanie identyfikatora sesji w URL-u
  - zagnieżdżanie w niewidocznych polach formularza
  - użycie ciasteczek (cookies)

Sesja (session)

- przechowuje dane pomiędzy kolejnymi żądaniami
- posiada strukturę tablicy asocjacyjnej
- może przechowywać obiekt dowolnego typu
- `session['user_id']=current_user.id`
- identyfikator sesji jest przechowywany w `cookie` i wysyłany przy każdym żądaniu
- różnorodne opcje przechowywania – pliki, baza danych, cookie
- domyślnie – cookie
- nie trzeba usuwać „umarłych” sesji

Ciasteczka (cookies)

- przechowywane w przeglądarce użytkownika
- przesyłane z każdym żądaniem
- posiadają strukturę tablicy asocjacyjnej
- mogą przechowywać wyłącznie łańcuchy

```
#set remember me token
cookies[:auth_token] = {
  :value => @session[:user].remember_token ,
  :expires => @session[:user].remember_token_expires
}
#find user by remember me token stored in cookie
user = User.find_by_remember_token(cookies[:auth_token])
```

flash

- ActionController::Flash
- pozwala na przekazywane tymczasowych obiektów pomiędzy akcjami
- wszystko co zostanie umieszczone w flashu, zostanie udostępnione następnej akcji a później wyczyszczone
- najczęściej wykorzystywany do ustawiania komunikatów w akcjach, po których następuje przekierowanie do innej akcji
- ustawianie wiadomości wygląda jak wstawianie ich do tablicy asocjacyjnej:

```
flash[:notice]='You have successfully logged in!'
flash[:error]='Login is invalid!'
```

Przykład wykorzystania flash-a

```
class WeblogController < ActionController::Base
  def create
    # save post
    flash[:notice] = "Successfully created post"
    redirect_to :action => "display", :params => {:id => post.id}
  end

  def display
    #flash is still set
  end
end
```

```
<% if flash[:notice] %>
  <div class="notice"><%= flash[:notice] %></div>
<% end %>
```

Używanie loggera do debugowania

- przeglądanie końca pliku:
  - *nix: tail -f log/development.log
  - radrails/aptana: opcja „tail” w menu kontekstowym
- użycie loggera – w kontrolerze:

```
logger.debug "user.id = #{user.id}"
logger.info "Starting process fubar..."
logger.warn "No results found for XYZ"
logger.error err.message
logger.fatal "Database down"
```

- inspekcja zmiennych: @var.inspect

http://maintainable.com/articles/rails_logging_tips

Rozszerzenia Firefox-a przydatne do debugowania

- FireBug
- WebDeveloper
- Live HTTP Headers