Routování

v Nette Framework

Jan „Panda“ Smitka
@jansmitka

28. 1. 2012

Co to vlastně routování je?

- obousměrný překlad mezi URL adresou a vnitřním požadavkem aplikace
  - **URL** → `Nette\Application\Request`
    požadavek klienta se mapuje na presenter a jeho parametry
  - `Nette\Application\Request` → **URL**
    presenter a parametry se mapují na URL – generování odkazů
- samostatná vrstva aplikace – nezávislé na zbytku aplikace

Služba `router`

- služba, která se stará o samotný překlad mezi URL a požadavkem aplikace
- jakákoliv třída implementující `Nette\Application\IRoute`
- výchozí implementací služby je `Nette\Application\RouteList`
- díky této službě je možné logiku routování kdykoliv snadno vyměnit

RouteList

- slouží jako kolekce rout (pravidel) – objektů implementujících `IRoute`
- typicky inicializace v `bootstrap.php`:

```
1  $router = $container->router;
2  $router[] = new Route('index.php', 'Homepage:default', Route::ONE_WAY);
3  $router[] = new Route('<presenter>/<action>[/<id>]', 'Homepage:default');
```

- přidání routy stejné, jako přidání prvku na konec pole
- může obsahovat jakýkoliv prvek typu `IRouter` – tedy i další `RouteList`

Vnořený RouteList

- přidání jako jakékoliv jiné routy:

```
1 $router[] = new RouteList();
2 $router[] = new RouteList('Admin');
```

- jediný nepovinný parametr konstruktoru přejímá název modulu
- velmi výhodné u modulárních aplikací – při inicializaci modulu se routy nemusí přidávat přímo do hlavního routeru, pouze stačí vytvořit vnořený `RouteList`
- routy v tomto vnořeném routeru již nemusí zohledňovat název modulu v parametrech

Hledání routy

- při vyhledávání vhodné routy, která se použije pro překlad, se v seznamu hledá sekvenčně
- použije se první routa, která odpovídá URL či požadavku
- **záleží tedy na pořadí!**
- obecnější routy na konec, specifické na začátek
- při ladění nám může pomoci routing debugger

Route

- představuje jedno routovací pravidlo
- určeno maskou URL a parametry:

```
$router[] = new Route('<presenter>/<action>[/<id>]', 'Homepage:default');
```

- ke správné funkci je potřeba zapnutý a nastavený `mod_rewrite`, případně podobný mechanismus – `try_files` u nginx, URL rewrite u IIS...

Tvar masky

- adresa relativní k document root aplikace (složka `www`):

  `'<presenter>/<action>[/<id>]'`

- adresa relativní k document root serveru:

  `'/<presenter>/<action>[/<id>]'`

- absolutní adresa včetně hostname:

  `'//<module>.example.com/<presenter>/<action>[/<id>]'`

Parametry v masce

`'<presenter>/<action>/<id>'`

- pojmenované parametry jsou v masce uzavřeny ve špičatých závorkách: např. `<id>`
- tyto parametry se pak předávají presenteru, případně view, pokud má definovaný parametr daného názvu:

```
1  public function renderDefault($id) {
2      // ...
3  }
```

- prohledávány jsou jak metody `action<action>()`, tak `render<view>()` – jejich signatura by měla být stejná
- speciální parametry: `module`, `presenter`, `action` – určují presenter a akci, která se má vykonat

Parametry v masce – validační výrazy

```
'<presenter page|article>/<action>/<id \d+>'
```

- u každého parametru lze určit regulární výraz, kterému má parametr odpovídat
- hodnota parametru se kontroluje při překladu v obou směrech

Nepovinné části adresy

```
'<presenter>/<action>[/<id>]'
'index[.html]'
```

- nepovinné části adresy se uzavírají do hranatých závorek
- nepovinné může být cokoliv: parametr, statický řetězec...
- při generování adresy snaha o co nejkratší URL – „vypustit, co se dá“
  - statický řetězec bude vždy vypuštěn, vykřičníkem před levou hranatou závorkou lze chování obrátit:

    ```
    'index[!.html]'
    ```
  - část s parametry, které nabývají výchozí hodnoty, bude také vypuštěna

Foo parametry

`'index.<? \.html?|\.php|>'`

- podobné nepovinným částem adresy
- specifikuje regulární výraz
- foo parametr bude vždy z adresy odstraněn

Výchozí hodnoty parametrů

- přímo v masce:

```
new Route('<presenter=Homepage>/<action=default>[/<id=5>]')
```

- druhý parametr konstruktoru:
  - pro akci presenteru je možné použít řetězec:

```
new Route('<presenter>/<action>[/<id>]', 'Homepage:default')
```

  - pole parametrů:

```
new Route('<presenter>/<action>[/<id>]', array(
    'presenter' => 'Homepage',
    'action' => 'default',
    'id' = 5
))
```

Transformace parametrů

- parametry je možné před předáním presenteru nebo před vytvořením URL adresy filtrovat pomocí funkcí:

```php
new Route('stranka/<id>', array(
    'id' => array(
        Route::VALUE => NULL,
        Route::FILTER_IN => function ($str) {
            // ...
            return $str;
        },
        Route::FILTER_OUT => function ($str) {
            // ...
            return $str;
        }
    )
))
```

`Route::VALUE` určuje výchozí hodnotu

`Route::FILTER_IN` specifikuje filtr při překladu z URL do parametrů

`Route::FILTER_OUT` specifikuje filtr při vytváření URL z parametrů

Překladová tabulka

- je možné použít také překladovou tabulku:

```
1  new Route('<presenter>/<action>[/<id>]', array(
2      'presenter' => array(
3          Route::FILTER_TABLE => array(
4              'stranka' => 'Page',
5              'clanek' => 'Article',
6              'produkt' => 'Product'
7          )
8      )
9  ))
```

SimpleRouter

- náhrada sadu `Route` v `RouteList`
- veškeré parametry jsou předávány v `GET`
- nevyžaduje `mod_rewrite`

Příklad použití

```
1  $container->router = new SimpleRouter('Homepage:default');
```

Kanonizace URL

- k jednomu objektu může být možné se dostat pomocí více URL, typicky `/` vs. `/index.php`
- to může mást vyhledávače – nutno řešit, ale Nette to vyřeší za nás
  1. proběhne normální routování, spustí se presenter
  2. v průběhu životního cyklu po zavolání `action<action>()` se pokusí vygenerovat URL pro aktuálně zpracovávaný požadavek
  3. pokud se adresy liší, proběhne přesměrování na novou adresu
- protože při generování odkazů se používá vždy první vyhovující routa, vede to k vytváření jednotných odkazů

Dotazy?

Díky za pozornost!