Symfony 1.4 - Sviluppo di base

Loris Tissino

11 maggio 2011

Introduzione

Quelle che seguono sono veloci indicazioni su come sviluppare un'applicazione con Symfony, e non pretendono minimamente di sostituire la lettura della documentazione presente su web nel sito web del progetto.

Anatomia di un'azione

Quando da un browser viene richiesta una certa azione (con un GET o un POST ad un determinato URL), vengono svolte le seguenti attività:

1. *routing*
   - verifica URL
   - controllo parametri
   - scelta dell'azione
2. *filtraggio*, prima dell'esecuzione
   - controlli vari, ad esempio:
     - l'utente è autenticato?
     - l'utente ha le credenziali adeguate?
3. *azione*, prima parte
   - valorizzazione variabili
   - creazione istanze
   - ulteriori controlli
   - invocazione di funzioni del modello
4. *lavoro del modello*
   - (diverse a seconda dei casi)
5. *azione*, seconda parte
   - valorizzazione variabili in base al risultato del lavoro del modello
   - scelta sul come procedere
     - *redirect* (richiesta al browser)?
     - *forward* (interna all'applicazione)?
     - resa con un *template* (default?) ed eventualmente *layout*?
6. *output*
   - impaginazione dei risultati con il *template* specifico
   - eventuale inclusione di *parziali*
   - eventuale definizione di contenuti per gli *slot*
7. *filtraggio*, finale
   - sostituzione di qualche parte dell'output, se necessario

Esempi di azione

Le varie azioni spesso riflettono URL del tipo

```
room/list
room/index
room/edit/id/3
room/show/id/3
```

Nel caso di `room/edit/id/3`, il modulo è *room*, l'azione è *edit* e *id=3* è il parametro fornito.

Verrà quindi eseguita la funzione `executeEdit` della classe `roomActions`.

Cosa fare in un'azione

All'interno del codice dell'azione si potrà:

- usare il parametro `$request` per sapere qual è il valore dei parametri

  ```
  $id = $request->getParameter('id');
  ```

- impostare, se lo si desidera, un template diverso da quello di default (il cui nome corrisponde a quello dell'azione)

  ```
  $this->setTemplate('editp1');
  ```

- impostare, se lo si desidera, un layout diverso da quello di default (che è quello generico per il modulo o per l'applicazione)

  ```
  $this->setLayout('popupwindow');
  ```

- impostare dei valori per qualsiasi variabile debba essere visualizzata nel template, eventualmente facendo riferimento a valori presenti in un file di configurazione

  ```
  $this->name=sfConfig::get('app_config_name', 'test');
  ```

- scrivere qualche nota nel file di log, a scopo di debug

  ```
  $this->logMessage('Ehi, sono nel modulo ' . $this->getModuleName(), 'info');
  ```

- ridirigere il browser verso qualche altra azione, qualche altro modulo, o verso una pagina di errore (l'URL del browser cambia)

  ```
  $this->forward('hello', 'index'); // modulo, azione
  $this->forward404();
  $this->forward404Unless($room=RoomPeer::retrieveByPk($id));
  ```

- ridirigere il flusso esecutivo verso una diversa funzione (l'URL del browser non cambia)

  ```
  $this->redirect('@forbidden');    // voce della tabella di routing
  ```

- rendere direttamente un testo

  ```
  return $this->renderText('The result is: 42');
  ```

- rendere direttamente un parziale

  ```
  return $this->renderPartial('room/details');
  ```

- impostare particolari intestazioni HTTP

```
$this->getContext()->getResponse()->setHttpHeader('Content-Type', 'image/jpeg');
```

- impostare un'informazione Flash per l'utente (di solito associato ad un forward)

```
$this->getUser()->setFlash('info', 'Impostazioni salvate');

/* nel template dell'azione o nel layout sarà necessario un codice come:

<?php if($sf_user->hasFlash('info')): ?>
  <?php echo $sf_user->getFlash('info') ?>
<?php endif ?>
*/
```

- impostare un valore di uscita

```
// uno di questi valori:
return sfView::SUCCESS;  // success view (default, non è necessario specificarlo)
return sfView::ERROR;    // error view
return sfView::ALERT;    // alert view
return sfView::NONE;     // view execution skipped
```