Miniguida a Symfony 1.4

Loris Tissino

29 aprile 2011

Introduzione

Questi appunti servono a dare delle indicazioni di massima su come usare il framework Symfony 1.4 per sviluppare un applicazione web MVC (model-view-controller) usando l'ORM Propel e un database MySQL.

Informazioni più dettagliate sui vari argomenti trattati si possono trovare nel sito del progetto Symfony.

Per progetti più a lungo termine, converrà tenere in considerazione Symfony 2.

Organizzazione degli esempi

Gli appunti fanno riferimento ad una applicazione generica, chiamata *php5webdevsf1*. Qualsiasi riferimento a questo nome andrà cambiato con quello alla propria applicazione. Il file `config/properties.ini` contiene il nome del progetto, che viene utilizzato nei commenti del codice autogenerato.

Gli esempi sono stati testati con un server web Apache 2 in una macchina Linux e con il programma XAMPP su una macchina Windows XP e su una macchina Windows 7. In linea di massima, con i necessari adattamenti, dovrebbero funzionare anche con altre configurazioni.

Versioni aggiornate di questo documento e di file di esempio dovrebbero essere disponibili all'indirizzo web

`http://loris.tissino.it/php5webdevsf1`

Cose a cui prestare attenzione

I file degli esempi sono codificati in utf8 e con i caratteri di fine riga secondo lo standard Linux/Unix. Sotto Microsoft Windows, sarà necessario usare un editor serio (ossia non *notepad*) per visualizzarli correttamente.

I file di configurazione con estensione `.yml` sono strutturati secondo lo standard YAML, che prevede una gerarchia delle voci basata sul numero di spazi all'inizio di ogni riga. Attenzione a non pasticciare.

La versione PDF di questo documento viene generata con un procedimento automatico che converte gli apici semplici in apici ricurvi. Questo potrebbe causare errori se si fanno operazioni di copia&incolla.

Configurazione server web

Apache2 sotto Ubuntu/Linux

Controllare di avere installato i pacchetti necessari:

```
sudo apt-get update
sudo apt-get install apache2 apache2-utils phpmyadmin \
  php5 php5-mysql php5-xsl php5-cli mysql-server
```

Scompattare il file del progetto nella directory `/var/php5webdevsf1`.

È consigliabile impostare un sito specifico per la propria applicazione. Ad esempio, in `/etc/apache2/sites-available` si può predisporre un file `php5webdevsf1` con il seguente contenuto:

```
<VirtualHost 127.0.0.1:80>
  ServerName localhost.local
  DocumentRoot "/var/php5webdevsf1/web"
  DirectoryIndex index.php
  <Directory "/var/php5webdevsf1/web">
    AllowOverride All
    Allow from All
  </Directory>
  php_flag display_errors on
  Alias /sf /var/php5webdevsf1/lib/vendor/symfony/data/web/sf
  <Directory "/var/php5webdevsf1/lib/vendor/symfony/data/web/sf">
    AllowOverride All
    Allow from All
  </Directory>
</VirtualHost>
```

Modificare i file `/etc/php5/apache2/php.ini` e `/etc/php5/cli/php.ini` per modificare la riga short_open_tags:

```
short_open_tags = Off
```

Impartire i comandi

```
sudo a2ensite php5webdevsf1
sudo service apache2 restart
```

per abilitare il sito e riavviare il server web.

Il file `symfony` deve essere reso eseguibile:

```
sudo chmod +x symfony
```

Può essere conveniente creare un link simbolico al file `symfony` in una directory compresa tra quelle elencate nella variabile PATH. Ad esempio:

```
sudo ln -s /var/php5webdevsf1/symfony /usr/local/bin/symfony
```

Controllare le impostazioni dello script `utilities/fixpermissions.sh` per vedere se rispondono alle proprie esigenze in merito ai permessi sui file e sulle directory, e poi eseguirlo: s utilities/fixpermissions.sh

XAMPP sotto Microsoft Windows

Dopo aver copiato la directory del progetto sotto `htdocs`, nel file `C:\xampp\apache\conf\httpd.conf` bisogna aggiungere alla fine una riga con il seguente valore:

```
Alias /sf "c:/xampp/htdocs/php5webdevsf1/lib/vendor/symfony/data/web/sf"
```

Nello stesso file, la voce `DocumentRoot` va cambiata in questo modo:

```
DocumentRoot "c:/xampp/htdocs/php5webdevsf1/web"
```

Nel file `c:/xampp/php/php.ini` bisogna aggiungere la riga

```
extension=php_xsl.dll
```

per abilitare le trasformazioni *xslt* di php e modificare la riga `short_open_tags`:

```
short_open_tags = Off
```

(attenzione al fatto che con Xampp la riga è ripetuta più volte).

Per poter eseguire php da riga di comando, bisogna aggiungere la directory dove è posto l'eseguibile (`c:\xampp\php`) nella variabile PATH (*Pannello di controllo*, *Sistema e sicurezza*, *Impostazioni di di sistema avanzate*, *Variabili d'ambiente*, *Variabili di sistema*). Potrebbe essere necessario un riavvio del calcolatore perché la modifica abbia effetto.

In alternativa, si dovrà specificare ogni volta il percorso completo dell'eseguibile php ad ogni invocazione del comando `symfony`:

```
\xampp\php\php.exe symfony
```

Test della configurazione

Chiamare con il browser la pagina `check_configuration.php` per controllare che tutto sia in ordine.

```
http://localhost/check_configuration.php
```

Chiamare lo stesso file anche da riga di comando:

```
php web/check_configuration.php
```

Database MySQL

Le operazioni seguenti possono essere svolte agevolmente tramite l'interfaccia web `phpmyadmin`.

Creazione database

Il primo passo sarà di creare i database per l'ambiente di sviluppo, quello di test e quello di produzione:

Tramite *phpMyAdmin* eseguire i seguenti comandi SQL:

```
CREATE DATABASE php5webdevsf1_dev DEFAULT CHARACTER SET utf8
  COLLATE utf8_general_ci;
CREATE DATABASE php5webdevsf1_test DEFAULT CHARACTER SET utf8
  COLLATE utf8_general_ci;
CREATE DATABASE php5webdevsf1_prod DEFAULT CHARACTER SET utf8
  COLLATE utf8_general_ci;
```

Creazione utente nel db

È necessario che ci sia un utente che possa accedere al database (si tratta dell'utente che rappresenta l'applicazione web per il database, non l'utente "umano" che usa l'applicazione web). Per semplicità, possiamo garantire al nostro utente tutti i diritti sui database che iniziano con il suo nome, che corrisponde al nome dell'applicazione:

Tramite *phpMyAdmin* (scheda *Privilegi*) possiamo creare un utente `php5webdevsf1` che da *localhost* possa accedere a MySQL. Ci si accerti di selezionare la voce *Concedi tutti i privilegi al nome con caratteri jolly (username_%)* nella sezione *Database per l'utente* e che non sia selezionato nessun privilegio globale.

Per la password ci si regoli come si preferisce. In genere raccomandarei di far generare una password casuale a *phpMyAdmin*.

File di configurazione per accesso al DB

Il file di configurazione per l'accesso al DB è `config/databases.yml`, che dovrebbe avere una struttura simile a quella presentata qui di seguito (salvo, con ogni probabilità, una password diversa).

Il meccanismo di configurazione è "a cascata": sotto *all* sono riportate le impostazioni generiche, che vengono riprese anche per le altre voci, a meno che non siano esplicitamente indicate le variazioni.

```
all:
  propel:
    class: sfPropelDatabase
    param:
      dsn: 'mysql:host=localhost;dbname=php5webdevsf1_dev'
      username: php5webdevsf1
      password: WAD4FDm4F7
      persistent: true
      encoding: utf8
      classname: DebugPDO
test:
  propel:
    param:
      dsn: 'mysql:host=localhost;dbname=php5webdevsf1_test'
prod:
  propel:
    param:
      dsn: 'mysql:host=localhost;dbname=php5webdevsf1_prod'
      classname: PropelPDO
```

Configurazione database per symfony

Per la generazione/modifica del file di configurazione per l'accesso al database da parte dell'applicazione si può, se lo si desidera, eseguire i comandi illustrati qui di seguito.

Saltare questa sezione se si è già prodotto il file di configurazione corretto.

Dalla riga di comando (shell sotto linux o prompt dei comandi sotto Windows) eseguire questi comandi per la configurazione:

```
symfony configure:database mysql:host=localhost;dbname=php5webdevsf1_dev php5webdevsf1 WAD4FDm4F7
symfony configure:database --env=prod mysql:host=localhost;dbname=php5webdevsf1_prod
symfony configure:database --env=test mysql:host=localhost;dbname=php5webdevsf1_test
```

Generazione applicazione frontend

L'applicazione di frontend viene generata con:

```
symfony generate:app --csrf-secret='kwalkohsa' frontend
```

dove il segreto CSRF serve a evitare attacchi di tipo *cross-site request forgery.*

Si noti che nella directory `web` vengono generati i file `frontend.php` e `frontend_dev.php`, rispettivamente per l'ambiente di produzione e per quello di sviluppo.

A questo punto, con il browser dovrebbe essere possibile accedere all'applicazione di frontend, nell'ambiente di sviluppo, puntando il browser all'indirizzo

```
http://localhost/frontend_dev.php
```

Predisposizione del modello

Creare o modificare il file `config/schema.yml` in modo che contenga queste righe:

```
propel:

  sf_guard_user_profile:
    _attributes:  { phpName: sfGuardUserProfile }
    user_id:
      type:             integer
```

Figure 1: Progetto creato

```
    foreignTable:     sf_guard_user
    foreignReference: id
    required:         true
    onUpdate:         cascade
    onDelete:         cascade
    primaryKey:       true
  first_name:   varchar(50)
  last_name:    varchar(50)
  email:        varchar(255)
  imported_at:  timestamp
  updated_at:   timestamp

room:
  _attributes:  { phpName: Room }
  id:           ~
  shortcut:     varchar(30)
  description:  varchar(255)
  internet_access: { type: boolean, default: false }
  created_at:   ~
  updated_at:   ~
```

Generazione del codice del modello

A partire dal modello, potremo generare il codice delle classi:

```
symfony propel:build-model
```

Otterremo così il codice delle classi seguenti:

```
lib/model/om/BaseRoom.php
lib/model/om/BasesfGuardUserProfilePeer.php
lib/model/om/BaseRoomPeer.php
lib/model/om/BasesfGuardUserProfile.php
lib/model/Room.php
lib/model/RoomPeer.php
lib/model/sfGuardUserProfile.php
lib/model/sfGuardUserProfilePeer.php
```

Generazione dei moduli (form)

Potremo poi generare il codice dei moduli per l'inserimento dati:

```
symfony propel:build-forms
```

Otterremo questi file:

```
lib/form/base/BaseRoomForm.class.php
lib/form/base/BasesfGuardUserProfileForm.class.php
lib/form/BaseFormPropel.class.php
lib/form/BaseForm.class.php
lib/form/sfGuardUserProfileForm.class.php
lib/form/RoomForm.class.php
```

Generazione dei filtri

Generiamo ora i filtri (presenti in alcune pagine web):

```
symfony propel:build-filters
```

Otterremo questi file:

```
lib/filter/sfGuardUserProfileFormFilter.class.php
lib/filter/base/BaseRoomFormFilter.class.php
lib/filter/base/BasesfGuardUserProfileFormFilter.class.php
lib/filter/RoomFormFilter.class.php
lib/filter/BaseFormFilterPropel.class.php
```

Generazione del codice SQL

Per generare il codice SQL (che servirà per la creazione delle tabelle) possiamo impartire il comando:

```
symfony propel:build-sql
```

I file vengono posti nella directory `data/lib`:

```
data/sql/plugins.sfGuardPlugin.lib.model.schema.sql
data/sql/lib.model.schema.sql
```

(Si noti che è nel file `config/propel.ini` che possono essere poste ulteriori configurazioni, come ad esempio il motore da usare per i database MySQL.)

Nota: sotto Windows/XAMPP, potrebbe essere necessario intervenire sul file

```
lib/vendor/symfony/lib/plugins/sfPropelPlugin/lib
/vendor/propel-generator/classes/propel/engine/
builder/sql/mysql/MysqlDDLBuilder.php
```

prima di eseguire il comando, con la modifica della riga 156, dove bisogna sostituire

```
$script .= "Type=$mysqlTableType";
```

con

```
$script .= "Engine=$mysqlTableType";
```

Creazione delle tabelle nel DB

Con i comandi

```
symfony propel:insert-sql
symfony propel:insert-sql --env=test
symfony propel:insert-sql --env=prod
```

vengono effettivamente create le tabelle nel database.

Predisposizione dei dati di prova

Nella directory `data/fixtures` si possono creare dei file contenenti dei dati di prova da (ri)caricare ogni volta che si vuole riazzerare la situazione.

I file vengono elaborati da php prima del loro uso effettivo, per cui si possono usare delle istruzioni per creare dati di prova numerosi, come nell'esempio seguente:

```
Room:
  mr1:
    shortcut: MR1
    description: Stanza riunioni 1
    internet_access: false
<?php for($i=1; $i<10; $i++): $ia = ($i % 2) ? 'true': 'false'?>
  room<?php echo $i ?>:
    shortcut: Lab<?php echo $i . "\n" ?>
    description: Laboratorio <?php echo $i . "\n" ?>
    internet_access: <?php echo $ia . "\n" ?>
<?php endfor ?>
```

Si può verificare quali dati verranno effettivamente caricati mediante il comando

```
php data/fixtures/010_rooms.yml
```

che porta ad un output come il seguente:

```
Room:
  mr1:
    shortcut: MR1
    description: Stanza riunioni 1
    internet_access: false
  room1:
    shortcut: Lab1
    description: Laboratorio 1
    internet_access: true
  room2:
    shortcut: Lab2
    description: Laboratorio 2
    internet_access: false
  room3:
    shortcut: Lab3
    description: Laboratorio 3
    internet_access: true
    ...
```

Caricamento dei dati di prova

Con il comando

```
symfony propel:data-load
```

vengono caricati i dati di prova.

Generazione di moduli indipendenti dal modello

Con il comando

```
symfony generate:module frontend easteregg
```

viene generato un modulo, chiamato *easteregg*, che può essere utilizzato indipendentemente dal modello dei dati.

Generazione dei moduli associati al modello

Con il comando

```
symfony propel:generate-module --with-show --non-verbose-templates frontend room Room
```

viene generato un modulo dell'applicazione di frontend, quello relativo alle stanze.

(L'opzione *non-verbose* è utile in fase di sviluppo, quando non è ancora definito quali devono essere i campi che fanno parte di un modulo inserimento dati.)

Personalizzazione dei moduli (form)

Nei moduli generati automaticamente sono possibili molte personalizzazioni. Ad esempio, per evitare che nel modulo compaiano data e ora di creazione e aggiornamento, è sufficiente agire sul file `lib/form/RoomForm.class.php` per disabilitare i due campi:

```
  public function configure()
  {
    unset(
      $this['created_at'],
      $this['updated_at']
      );
  }
```

Personalizzazione dei template

I template possono essere personalizzati come meglio si desidera. Ad esempio, per togliere le colonne relative a creazione e aggiornamento sarà sufficiente agire sulle rispettive righe del file `indexSuccess.php` presente nella directory `apps/frontend/modules/room/templates` (sia nell'intestazione della tabella, sia nel corpo).

Inoltre, è possibile utilizzare i cosiddetti *parziali*, in modo da sfruttare in più template lo stesso codice html.

```
<td><?php include_partial('internet', array('active'=>$Room->getInternetAccess())) ?></td>
```

Il parziale (in questo caso, il nome è `_internet.php`) potrebbe essere qualcosa di questo tipo:

```
<?php if($active): ?>
  ATTIVO
<?php else: ?>
  NON ATTIVO
<?php endif ?>
```

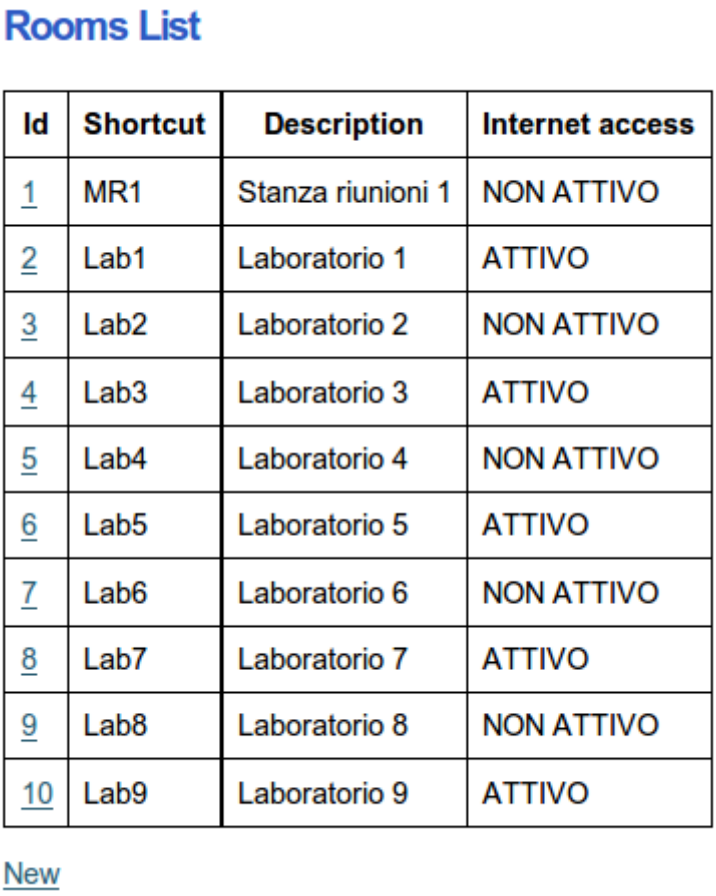
In questo modo si dovrebbe ottenere un risultato simile quello mostrato in figura 2.

Generazione applicazione di backend

L'applicazione di backend può essere generata con il comando

```
symfony generate:app --csrf-secret='44gifp6crd2' backend
```

dove, come per l'applicazione di backend, si potrà scegliere un adeguato valore per la protezione dei moduli da attacchi di tipo csrf.



Rooms List

Id	Shortcut	Description	Internet access
1	MR1	Stanza riunioni 1	NON ATTIVO
2	Lab1	Laboratorio 1	ATTIVO
3	Lab2	Laboratorio 2	NON ATTIVO
4	Lab3	Laboratorio 3	ATTIVO
5	Lab4	Laboratorio 4	NON ATTIVO
6	Lab5	Laboratorio 5	ATTIVO
7	Lab6	Laboratorio 6	NON ATTIVO
8	Lab7	Laboratorio 7	ATTIVO
9	Lab8	Laboratorio 8	NON ATTIVO
10	Lab9	Laboratorio 9	ATTIVO

New

Figure 2: Lista aule

Generazione dei moduli di backend

I moduli di backend possono essere generati con comandi simili al seguente:

```
symfony propel:generate-admin backend room
```

I moduli di backend servono per una gestione più grezza della parte amministrativa dell'applicazione (operazioni CRUD), e consentono l'impostazione di filtri, la predisposizione di operazioni batch, ecc.

Il codice dei moduli generati in questo modo viene memorizzato nella cache dell'applicazione e viene gestito, tipicamente, solo attraverso il file di configurazione `configurator.yml`.

Generazione di task

Un task può essere generato con un comando simile al seguente:

```
symfony generate:task php5webdevsf1:internet-disable \
  --use-database=propel \
  --brief-description="Disables internet access for all labs"
```

I task sono comodi per le operazioni di routine, da richiamare ad esempio con crontab.

Codice di un task

I task vengono eseguiti senza alcuna autenticazione e possono operare su tutto il database.

Un esempio di codice minimale che opera su tutte le aule per disabilitare l'accesso a internet potrebbe essere il seguente, da aggiungere al codice autogenerato:

```
// add your code here

$rooms=RoomPeer::doSelect(new Criteria());
foreach($rooms as $room)
{
  if($room->getInternetAccess())
  {
    $room
    ->setInternetAccess(false)
    ->save()
    ;
    $this->logSection('internet-', $room->getDescription(), null, 'INFO');
  }
  else
  {
    $this->logSection('internet=', $room->getDescription(), null, 'COMMENT');
  }
}
```

Esecuzione di un task

Il task può essere eseguito dalla riga di comando semplicemente invocandolo con il suo namespace e con eventuali parametri richiesti:

```
symfony php5webdevsf1:internet-disable
```

```
loris@uczen:/var/php5webdevsf1$ symfony php5webdevsf1:internet-disable
>> internet= Stanza riunioni 1
>> internet- Laboratorio 1
>> internet= Laboratorio 2
>> internet- Laboratorio 3
>> internet= Laboratorio 4
>> internet- Laboratorio 5
>> internet= Laboratorio 6
>> internet- Laboratorio 7
>> internet= Laboratorio 8
>> internet- Laboratorio 9
```

Figure 3: Esecuzione di un task

Lavoro sul modello

Per come sono state impostate le cose, nel momento in cui viene invocata l'azione *setInternetAccess()* viene solo cambiato il valore nel database. Se si vuole che venga fatto dell'altro, sarà necessario lavorare sul codice della classe `Room`.

Ad esempio, se ci fosse una classe astratta `Firewall` con un metodo *enableInternetAccess()* potremmo scrivere un codice simile al seguente:

```
class Room extends BaseRoom {

  // ...
  public function setInternetAccess($v)
  {
    Firewall::enableInternetAccess($this->getShortcut(), $v);
    parent::setInternetAccess($v);
  }
```

... ma ovviamente sono possibili mille altre soluzioni. Qui inizia il divertimento!