

第 14 章 - 生成器

很多程序基于存储在数据库里的数据并提供访问这些数据的界面。symfony 能够自动完成根据 Propel 对象生成数据处理模块这样的重复任务。如果模型定义的好，symfony 甚至可以自动生成整个网站后台。本章将会介绍 symfony 的两种生成器：脚手架生成器和管理生成器。其中后者依赖于一个特别的语法复杂的配置文件，所以这一章的大部分篇幅会用来介绍管理生成器的各种用法。

基于模型生成代码

在 web 应用程序里，数据访问操作可以归结为以下几类：

- 新增 (Creation) 一条记录
- 取得 (Retrieval) 记录
- 更新 (Update) 一条记录 (并且修改它的字段)
- 删除 (Deletion) 一条记录

这些操作很常见，它们有一个专门的缩写：CRUD。很多页面都可以简化成其中之一。例如，在论坛程序里，最新帖子列表就是一个取得记录的过程，回帖子是一个新增过程。

针对一个表的 CRUD 操作制作基本的动作 (action) 和模板在 web 程序里会经常出现。在 symfony 里，模型层包含的信息足够生成 CRUD 操作代码的需要，这样可以加快早期的后台界面开发。

所有的基于模型的代码生成任务都会建立整个模块，只要通过类似下面的一行 symfony 命令就可以完成：

```
> symfony <任务名> <应用程序名> <模块名> <类名>
```

代码生成任务包括 propel-init-crud、propel-generate-crud 和 propel-init-admin。

脚手架与管理界面

开发应用程序的过程中，代码生成有两种不同的用途：

- 脚手架是给定表 CRUD 操作所需的基本结构（动作与模板）。它的代码是最小化的，因为它需要成为后续开发的指导。它是起步的基础，经过修改后才能满足你的逻辑与表现的需求。脚手架大多用在开发阶段，用来提供数据库的 web 访问界面，建立一个原型，或者以此为基础制作一个与某个表相关的模块。

- 管理界面是专门用于数据处理的界面，多用于后台管理。管理界面与脚手架的不同点是它的代码不是用来手动修改的。它们可以被定制，扩展或者通过配置或继承进行装配。它们的外观很重要，它需要有排序，分页，还有过滤功能。管理界面可以作为软件的成品交给客户。

symfony 命令行用 crud 代表脚手架，用 admin 代表管理界面。

初始化或生成代码

symfony 有两种生成代码的方式：通过继承(init)或者代码生成(generate)。

你可以初始化一个模块，也就是建立空的继承自框架的类。这样可以避免动作(action)和模板的 PHP 代码被修改。如果你的数据结构还没最终确定或者你只需要一个快速的数据库接口来操作数据，这个功能很有用。运行时执行的代码不在你的应用程序里，而是在缓存里。这类生成任务的命令行任务名以 propel-init-开头。

初始化的动作(action)代码是空的。例如，一个初始化的 article 模块的代码可能会是这样：

```
class articleActions extends autoarticleActions
{
}
```

另一方面，你也可以生成动作(action)和模板的代码，这样可以修改它们。这样生成的模块不依赖于框架，并且不能被配置文件修改。这种生成任务的命令行任务名以 propel-generate-开头。

由于脚手架是后续开发的基础，所以最好生成一个脚手架。另外，管理界面需要能够通过配置文件方便的修改，模型变化的情况下也要能够使用。所以管理界面只能够初始化。

数据模型的例子

本章的所有演示 symfony 生成器功能的例子都基于这个简单的例子，这个例子可能会让你回想起第 8 章。这就是那个有名的博客应用程序，包含 Article 和 Comment 两个类。例 14-1 是它的设计(schema)。

例 14-1 - 博客应用程序的 schema.yml 文件

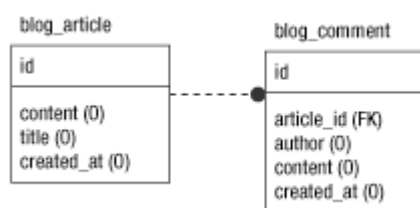
```
propel:
  blog_article:
    _attributes: { phpName: Article }
    id:
```

```

    title:          varchar(255)
    content:         longvarchar
    created_at:
blog_comment:
  _attributes: { phpName: Comment }
  id:
  article_id:
  author:          varchar(255)
  content:         longvarchar
  created_at:

```

图 14-1 - 例子的数据模型



代码生成并不会对设计(schema)的建立有什么特别的要求。symfony 会使用已有的设计(schema)，解释它的属性并生成脚手架或管理界面。

TIP 学习本章要达到最好的效果，你需要跟着这些例子去做。如果你按照这些例子中的每一个步骤去做，你会更好的理解 symfony 生成的代码以及它们的作用。所以建议你跟我们一起从刚才的这个例子做起，在一个数据库里面建立 blog_article 和 blog_comment 两个表，输入一些测试用的数据。

脚手架

脚手架在开发初期很有用。只要一条简单的命令，symfony 就能根据某个表的信息建立整个模块。

生成脚手架

根据 Article 模型类生成 article 模块，输入下面的命令：

```
> symfony propel-generate-crud myapp article Article
```

symfony 会读取 schema.yml 里 Article 类的定义并根据这些定义在 myapp/modules/article/ 目录建立一些模板和动作(action)。

生成的模块包括三个视图。list 视图，它是默认的视图，在浏览 http://localhost/myapp_dev.php/article_dev.php/article 的时候会显示 blog_article 表的记录，如图 14-2 所示。

图 14-2 - article 模块的 list 视图

article

Id	Title	Content	Created at
1	Welcome to the symfony weblog!	This is the first post of this weblog. Honestly, it is just a test to check if it works fine. Please comment it as much as you like.	2006-11-12 20:20:25
2	Life is beautiful	The purpose of a weblog is usually to talk about one's mood. Mine is great today. How is yours?	2006-11-12 20:20:25

create

点击文章 id 会显示 show 视图。这个页面显示的是这个记录的详细情况，如图 14-3。

图 14-3 - article 模块的 show 视图

Id: 1

Title: Welcome to the symfony weblog!

Content: This is the first post of this weblog. Honestly, it is just a test to check if it works fine. Please comment it as much as you like.

Created at: 2006-11-12 20:20:25

edit list

点击 edit 链接可以修改这篇文章，或者在 list 视图点击 create 链接新增一篇文章，会显示 edit 视图，如图 14-4 所示。

用这个模块，你可以新增文章，也可以修改或者删除已有的文章。生成的代码是未来开发的良好基础。例 14-2 列出了生成的新模块的动作 (action) 和模板代码

图 14-4 - article 模块的 edit 视图

Title: Welcome to the symfony

Content: This is the first post of this weblog. Honestly, it is just a test to check if it works fine. Please comment it as

save

deletecancel

例 14-2 - 生成的 CRUD 元素，在 myapp/modules/article/ 目录下

```
// 在 actions/actions.class.php 文件里
index // 转到下面的 list 动作
```

```

list           // 显示表里面的所有记录
show           // 显示一个记录的所有字段
edit           // 显示一个修改一条记录的表单
update        // 被 edit 动作调用的动作
delete        // 删除一条记录
create         // 新增一条记录

// 在 templates/ 目录下
editSuccess.php // 记录修改表单(edit 视图)
listSuccess.php // 显示所有的记录 (list 视图)
showSuccess.php // 记录详情 (show 视图)

```

这些动作和模板的逻辑很简单明白，把它们列出来就能说明一切。例 14-3 里是一部分生成的动作类的代码。

例 14-3 - 生成的动作类，位于
myapp/modules/article/actions/actions.class.php

```

class articleActions extends sfActions
{
    public function executeIndex()
    {
        return $this->forward('article', 'list');
    }

    public function executeList()
    {
        $this->articles = ArticlePeer::doSelect(new Criteria());
    }

    public function executeShow()
    {
        $this->article = ArticlePeer::retrieveByPk($this->getRequestParameter('id'));
        $this->forward404Unless($this->article);
    }
    ...
}

```

按照你的需求修改生成的代码，重复对所有需要交互的表进行 CRUD 生成，这样你就有了一个可以工作的基本的应用程序了。生成脚手架大大加快了开发速度，让 symfony 来为你干脏活，你只要专注与界面还有细节。

初始化脚手架

初始化一个脚手架在你需要检查是否能够访问数据库里的数据的时候很有用。它建立起来很快，一旦你确定一切工作正常，删除它也很快。

初始化一个 Propel 脚手架，它将建立一个负责处理 Article 模型类的数据的名叫 article 的模块，输入下面的命令：

```
> symfony propel-init-crud myapp article Article
```

你可以通过默认的动作(action)访问 list 视图：

```
http://localhost/myapp_dev.php/article
```

结果页面和生成的脚手架完全一样。你可以把它们作为数据库的简单 web 界面。

如果你去看新建立的 article 模块的 action.class.php 文件，你会发现他是空的：所有的东西都是继承自自动生成的类。模板也一样，templates/ 目录里面没有模板文件。初始化的动作和模板后面的代码与生成的脚手架的代码与模板一样，不过它们是放在应用程序缓存里 (myproject/cache/myapp/prod/module/autoArticle/)。

在应用程序的开发过程中，开发者初始化脚手架来处理数据，而不去管界面。初始化的代码不是用来定制的；初始化的脚手架可以看作 PHPmyadmin 的简单替代品来管理数据。

管理界面

根据由 schema.yml 文件生成的模型类，symfony 可以生成更加先进，更适合你的应用程序的后台模块。完全可以仅仅使用生成的管理界面制作整个网站后台。本节例子将在 backend 应用程序中增加管理模块。如果你的项目没有 backend 应用程序，请用 init-app 任务建立 backend 应用程序的框架：

```
> symfony init-app backend
```

管理界面模块通过一个叫 generator.yml 的特殊配置文件的设置来表现模型，通过这个文件可以完全控制生成的组件还有外观。之前介绍的模块机制（布局，验证，路由，自定义配置，自动载入，等）也可以在这些生成的管理模块中使用。你还可以重写生成的动作或者模板来给生成的管理界面增加你自己的功能，不过修改 generator.yml 就能够达到大多数的需求，只有需求很特殊的时候才使用 PHP 代码。

初始化管理界面模块

在 symfony 里建立管理界面是以模块为单位的。使用 propel-init-admin 任务生成基于 Propel 对象的模块，这与初始化脚手架的语法类似：

```
> symfony propel-init-admin backend article Article
```

这条命令会在 backend 应用程序里根据 Article 类的定义新建一个 article 模块，可以通过下面的网址访问：

<http://localhost/backend.php/article>

生成的模块的界面如图 14-5、14-6 所示，这样专业的界面，直接用在商业应用程序里也没什么不妥。

图 14-5 - backend 应用程序的 article 模块的 list 视图

article list

Id	Title	Content	Created at
1	Welcome to the symfony weblog!	This is the first post of this weblog. Honestly, it is just a test to check if it works fine. Please comment it as much as you like.	December 1, 2006 1:17 PM
2	Life is beautiful	The purpose of a weblog is usually to talk about one's mood. Mine is great today. How is yours?	December 1, 2006 1:17 PM
2 results			

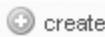



图 14-6 - backend 应用程序的 article 模块的 edit 视图




edit article


Title:

Content:

Created at:



 list |  save |  save and add

 delete

现在脚手架与管理界面的外观看上去并没有什么太大的差别，不过管理界面的可配置性可以使你不用写一行 PHP 代码就能够给这个基本布局增加很多功能。

NOTE 管理界面模块只能初始化（不能生成）。

浏览生成的代码

article 管理界面模块的代码，位于 apps/backend/modules/article/ 目录，由于刚刚初始化，还是空的。最好的查看生成代码的方法是在浏览器里访问它，然后去看 cache/ 目录的内容。例 14-4 列出了缓存中生成的动作与模板。

例 14-4 - 生成的管理界面元素，位于 cache/backend/ENV/modules/article/

```
// 位于 actions/actions.class.php
create          // 转到 edit
delete          // 删除一条记录
edit            // 显示修改记录的表单
                // 并且处理提交的表单
index           // 转到 list
list            // 显示表里的所有记录
save            // 转到 edit

// 位于 templates/
_edit_actions.php
_edit_footer.php
_edit_form.php
_edit_header.php
_edit_messages.php
_filters.php
_list.php
_list_actions.php
_list_footer.php
_list_header.php
_list_messages.php
_list_td_actions.php
_list_td_stacked.php
_list_td_tabular.php
_list_th_stacked.php
_list_th_tabular.php
editSuccess.php
listSuccess.php
```

从这里可以看出生成的管理界面模块主要由 edit 和 list 两个视图组成。如果你去看它们的代码，你会发现它们非常模块化，可读性强，并且容易扩展。

generator.yml 配置文件介绍

脚手架与管理界面的主要不同（除了管理界面不包括 show 动作）是管理界面依赖于 generator.yml YAML 配置文件里的参数。要看新建的管理界面模块的默认配置文件，打开位于 backend/modules/article/config/ 的 generator.yml 文

件，文件内容如例 14-5 所示。

例 14-5 - 默认的生成器配置，位于
backend/modules/article/config/generator.yml

```
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:   Article
    theme:         default
```

这个配置足够生成一个基本的管理界面。所有的定制参数都要加在 param 键下的 theme 这一行之后（也就是说所有在 generator.yml 文件里增加的内容至少以 4 个空格开头）。例 14-6 是一个典型的定制后的 generator.yml 文件。

例 14-6 - 典型的完全定制的生成器配置

```
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:   Article
    theme:         default

  fields:
    author_id:     { name: Article author }

  list:
    title:         List of all articles
    display:       [title, author_id, category_id]
    fields:
      published_on: { params: date_format=' dd/MM/yy' }
      layout:       stacked
      params:       |
        %%is_published%%<strong>%%=title%%</strong><br /><em>by %
%author%%
        in %%category%% (%%published_on%%)</em><p>%%content_summary%
%</p>
    filters:       [title, category_id, author_id, is_published]
    max_per_page:  2

  edit:
    title:         Editing article "%%title%%"
    display:
      "Post":      [title, category_id, content]
      "Workflow":  [author_id, is_published, created_on]
```

```

    fields:
      category_id: { params: disabled=true }
      is_published: { type: plain}
      created_on:   { type: plain, params: date_format='dd/MM/yy' }
      author_id:    { params: size=5 include_custom=>> Choose an
author << }
      published_on: { credentials:  }
      content:      { params: rich=true
tinymce_options=height:150 }

```

接下来将会详细解释这个配置文件里用到的所有参数。

生成器配置

生成器配置文件功能很强，可以用多种方式改变生成的管理界面。但是强大的功能需要付出代价：它的语法描述很长，难于阅读和学习，所以本章是整本书最长的章节之一。symfony 网站上提供了一个可以帮助你学习生成器配置的好东西：管理界面简要参考，如图 14-7 所示。下载网址(<http://www.symfony-project.com/uploads/assets/sfAdminGeneratorRefCard.pdf>)，建议在阅读本章的时候把它打印出来放在手边。

本节的例子会调整 article 管理界面模块还有基于 Comment 类定义的 comment 管理界面模块。通过 propel-init-admin 建立后者：

```
> symfony propel-init-admin backend comment Comment
```

图 14-7 - 管理界面简要参考

[illegible]

字段

默认情况，list 视图与 edit 视图的字段与 schema.yml 里定义的一样。在 generator.yml 文件里，你可以选择显示哪些字段，哪些隐藏，还有加入你自己的字段——甚至是与对象模型没有直接关系的字段。

字段设置

管理界面生成器会为 `schema.yml` 里定义的每个字段生成一个 `field`。在 `field` 键下，你可以修改每个字段如何显示、格式如何等。如 例 14-7 里为 `title` 字段定义了一个自定义的标签和输入控件类型，为 `content` 字段增加了一个提示。接下来将介绍这些参数如何工作。

例 14-7 - 给一个字段设置自定义标签

```
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:  Article
    theme:        default
```

```

    fields:
      title:          { name: Article Title, type: textarea_tag,
params: class=foo }
      content:        { name: Body, help: Fill in the article body }

```

除了这个针对所有视图的默认字段定义，你还可以在指定的视图里（list 与 edit）重写字段设置，如例 14-8 所示。

例 14-8 - 为每个视图重写字段设置

```

generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:  Article
    theme:        default

  fields:
    title:        { name: Article Title }
    content:       { name: Body }

  list:
    fields:
      title:       { name: Title }

  edit:
    fields:
      content:      { name: Body of the article }

```

基本原则是：所有 fields 键下的模块全局设置可以在特定的视图里（list 与 edit）重写。

增加显示的字段

在 fields 部分定义的字段可以在各视图中显示、隐藏、排列或者按照不同的方式分组。display 键就用于这个目的。例如，使用例 14-9 的代码安排 comment 模块的字段。

例 14-9 - 选择显示的字段，位于 modules/comment/config/generator.yml

```

generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:  Comment
    theme:        default

```

```

fields:
  article_id:      { name: Article }
  created_at:      { name: Published on }
  content:         { name: Body }

list:
  display:         [id, article_id, content]

edit:
  display:
    NONE:          [article_id]
    Editable:      [author, content, created_at]

```

list 视图会显示 3 个字段，如图 14-8 所示，edit 表单将会显示分成两组的 4 个字段，如图 14-9 所示。

图 14-8 - comment 模块的 list 视图的自定义字段设置

comment list

Id Article Body		
1	1	Well, if this comment displays, it means that your weblog does work...
2	1	Thank you for your feedback. It really helps to see people understanding you.
3	2	Mine is not bad either. I'd like to see more deers, though.
4	2	How can you be so positive? There are so many subjects to worry about out there!
5	2	Why is it always like that, people quarrelling as soon as they have room to express themselves?
5 results		


 create

图 14-9 - 分组显示 comment 模块的 edit 视图里的字段

edit comment

Article:	1 
Editable	
Author:	<input type="text" value="Anonymous"/>
Body:	<div>Well, if this comment displays, it means that your weblog does work...</div>
Published on:	<input type="text" value="12/1/06"/> 

 list |  save |  save and add |  delete

有两种方式设置 display:

- 选择要显示的字段，按照出现的顺序排列，放在简单的数组里——如前一个 list 视图。
- 分组显示，使用一个关联数组，组的名字作为键名，或者使用 NONE 代表没有名字的组。值仍然是有序的字段名字的数组。

TIP 默认情况，主键字段不会在任何一个视图里出现。

自定义字段

事实上，generator.yml 里定义的字段甚至可以不必要对应 schema 里定义的字段。只要相关的类提供一个自定的 getter，它就可以在 list 视图作为一个字段显示；如果同时有 getter 和 setter，它也可以用在 edit 视图里作为一个字段显示。例如，你可以用 getNbComments() 扩展 Article 模型，如例 14-10。

例 14-10 - 在模块中添加自定义的获取方法，位于 lib/model/Article.class.php

```
public function getNbComments()
{
    return $this->countComments();
}
```

那么 nb_comments 就可以作为一个生成的模块的字段（注意 getter 使用的是驼峰命名法 camelCase 版本的字段名作为方法名），如例 14-11。

例 14-11 - 自定义获取方法为管理界面模块提供额外的字段， 位于 backend/modules/article/config/generator.yml

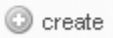
```
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:   Article
    theme:         default

  list:
    display:       [id, title, nb_comments, created_at]
```

article 模块的 list 视图的结果如图 14-10 所示。

图 14-10 - article 模块的 list 视图的自定义字段

article list

Id Title		Nb comments Created at	
1	Welcome to the symfony weblog!	2	December 1, 2006 1:17 PM
2	Life is beautiful	3	December 1, 2006 1:17 PM
2 results			
			

自定义字段除了返回原始数据之外还可以返回 HTML 代码。例如，你可以像例 14-12 那样用 `getArticleLink()` 方法扩展 `Comment` 类。

例 14-12 - 添加自定义的获取方法返回 HTML 代码， 位于 lib/model/Comment.class.php

```
public function getArticleLink()
{
    return link_to($this->getArticle()->getTitle(), 'article/edit?id=' .
    $this->getArticleId());
}
```

你可以在 `comment/list` 视图里使用与例 14-11 里相同的语法把这个新 getter 作为自定义字段。请看例 14-13，还有图 14-11 里的结果， getter 返回的 HTML 代码（到文章的超链接）出现在第 2 个字段而不是原来的主键。

例 14-13 - 自定义的返回 HTML 代码的 getter 也可以作为自定义字段， 位于 modules/comment/config/generator.yml

```

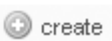
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:  Comment
    theme:        default

  list:
    display:      [id, article_link, content]

```

图 14-11 - comment 模块的 list 视图的自定义字段

comment list

Id	Article link	Body
1	Welcome to the symfony weblog!	Well, if this comment displays, it means that your weblog does work...
2	Welcome to the symfony weblog!	Thank you for your feedback. It really helps to see people understanding you.
3	Life is beautiful	Mine is not bad either. I'd like to see more deers, though.
4	Life is beautiful	How can you be so positive? There are so many subjects to worry about out there!
5	Life is beautiful	Why is it always like that, people quarrelling as soon as they have room to express themselves?
5 results		
		

局部模板字段

模型中的代码必须与表现无关。之前的 `getArticleLink()` 这个方法没有遵循这个层分离原则，因为它的代码里包含了视图代码。要用正确的方法达到相同的目的，必须要把 HTML 输出的代码放在一个自定义的局部模板里。幸运的是，只要在字段名前加上一个下划线，管理界面生成器就会认为这是一个局部模板字段。这样，例 14-13 中的 `generator.yml` 需要被改成例 14-14 中列出来的内容。

例 14-14 - 局部模板能用于附加列 — 使用 `_` 前缀

```

list:
  display:      [id, _article_link, created_at]

```

为了让这个配置能够正常工作，还需要在 `modules/comment/templates/` 目录里面增加一个名为 `_article_link.php` 的局部模板，内容如例 14-15 所示。

例 14-15 - 为 article 模块的 edit 视图自定义局部模板的例子，位于 `modules/comment/templates/_article_link.php`


```
<?php echo link_to($comment->getArticle()->getTitle(), 'article/edit?id='.$comment->getArticleId()) ?>
```

请注意局部模板字段的局部模板中可以通过以类命名的变量（此例中是 `$comment`）访问到当前对象。例如，一个为名为 `UserGroup` 的类建立的模块，在它的局部模板里可以通过 `$user_group` 变量访问到当前对象。

上面例子的结果与图 14-11 中的一样，不过这样作符合层分离原则。如果你习惯遵循这一原则，你的应用程序会更加容易维护。

给一个局部模板字段自定义参数的方法与普通字段完全一样——在 `field` 键下作出定义。只是要去掉前面的下划线（`_`）——请看例 14-16。

例 14-16 - 局部模板的属性可以在 `field` 键下定义

```
fields:
  article_link: { name: Article }
```

如果局部模板包含复杂的逻辑，你可以用组件取代它。把 `_` 前缀改成 `~`，你就定义了一个组件字段，如例 14-17 所示。

例 14-17 - 使用 `~` 前缀可以定义组件字段

```
...
list:
  display: [id, ~article_link, created_at]
```

在生成的模板里，这个例子的结果会是一个到当前模块的 `articleLink` 组件的调用。

NOTE 自定义字段与局部模板字段可以用在 `list` 视图与 `edit` 视图，还有过滤器里。如果在几个视图里使用同一个局部模板，可以从 `$type` 变量里取得当前环境（`list`，`edit` 或者 `filter`）。

视图定制

如果要修改 `edit` 与 `list` 视图的外观，你可能会尝试修改模板。但是因为他们自动生成的，这并不是一个好主意。你应该使用 `generator.yml` 配置文件，因为它几乎可以完成每件事而不需要太多修改。

修改视图标题

除了自定义字段，`list` 与 `edit` 页面还可以自定义页面标题。例如，如果想自定义 `article` 模块视图的标题，可以用例 14-18 里的配置。`edit` 视图的结果如图 14-12 所示。

例 14-18 - 为每个视图设置自定义标题，位于
backend/modules/article/config/generator.yml

```
list:
  title:      List of Articles
  ...

edit:
  title:      Body of article %%title%%
  display:    [content]
```

图 14-12 - 自定义 article 模块的 edit 视图的标题

Body of article Welcome to the symfony weblog!

Content:

This is the first post of this weblog.
Honestly, it is just a test to check if it
works fine. Please comment it as much
as you like.

list

save

save and add

delete

默认情况会用类名作为标题，很多时候这么作就够了——不过这需要你的类命名比较清楚明白。

TIP generator.yml 里的字符串值里，字段的值可以通过用 %%字段名%% 的方式取得。

增加提示

在 list 与 edit 视图里，可以增加用来描述字段的提示。例如，comment 模块的 edit 视图的 article_id 字段，在 fields 定义下面增加一个 help 属性，如例 14-19 所示。结果见图 14-13。

例 14-19 - 给 edit 视图设置提示，位于
modules/comment/config/generator.yml

```
edit:
  fields:
    ...
    article_id:  { help: The current comment relates to this
article }
```

图 14-13 - comment 模块的 edit 视图里的提示

edit comment

Article:

1

The current comment relates to this article

在 list 视图里，提示会显示在表头里，在 edit 视图里，它们会在输入框下面出现。

修改日期格式

可以用 `date_format` 参数指定日期的显示格式，如例 14-20 所示。

例 14-20 - 在 list 视图里设置日期显示格式

```
list:
  fields:
    created_at:      { name: Published, params:
date_format=' dd/MM' }
```

它的参数与之前介绍过的 `format_date()` 辅助方法一样。

SIDEBAR 管理界面模板是 I18N(国际化)的

所有生成的模板里的文字都经过了自动的国际化处理（例如，在外边包了一个 `__()` 辅助函数调用）。这也就是说，要翻译生成的管理界面很容易，只要在 `apps/myapp/i18n/` 目录的 XLIFF 文件里增加词语的翻译就可以了，详见之前章节的介绍。

list 视图相关的定制

list 视图可以以表格的形式显示记录的细节，或者把所有的细节放在一行显示。它还可以包含过滤器，翻页，还有排序功能。这些功能可以通过配置文件修改，下面将会介绍这些功能。

修改布局

默认情况，list 视图到 edit 视图的链接放在主键字段。如果你回去看图 14-11，你会发现留言列表的 `id` 字段不仅会显示每个留言的主键值，而且会有一个可以让用户访问 edit 视图的超链接。

如果你想要把修改记录的超链接放到其他字段，只要在 `display` 键的这个字段的名字前加一个等于号(=)。例 14-21 告诉我们如何把 `id` 从留言 list 里去掉，然后把它上面的超链接放到 `content` 字段上。图 14-14 是这个配置的运行结果

的截图。


例 14-21 - 改变 list 视图里到 edit 视图超链接的位置，位于 modules/comment/config/generator.yml

```
list:
  display: [article_link, =content]
```

图 14-14 - 移动 comment 模块的 list 视图里的到 edit 的超链接到其他字段

comment list

Article	Body
Welcome to the symfony weblog!	Well, if this comment displays, it means that your weblog does work...
Welcome to the symfony weblog!	Thank you for your feedback. It really helps to see people understanding you.
Life is beautiful	Mine is not bad either. I'd like to see more deers, though.
Life is beautiful	How can you be so positive? There are so many subjects to worry about out there!
Life is beautiful	Why is it always like that, people quarrelling as soon as they have room to express themselves?
5 results	

 create

默认情况，list 视图使用 tabular 布局，这种布局里字段显示成表格的列，前面的图里面就是这种布局。不过你也可以使用 stacked 布局把字段集中到一个字符串里，整个表格只有一列。如果你选择 stacked 布局，你需要设置 params 键的值来定义每一行要怎么显示。例如， 例 14-22 给 comment 模块的 list 视图定义了一个 stacked 布局，结果如图 14-15。

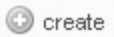
例 14-22 - 在 list 视图里使用 stacked 布局，位于 modules/comment/config/generator.yml

```
list:
  layout: stacked
  params: |
    %=content%% <br />
    (sent by %%author%% on %%created_at%% about %%article_link%%)
  display: [created_at, author, content]
```

图 14-15 - comment 模块的 list 视图里使用 stacked 布局

comment list

Published on	Author	Body
Well, if this comment displays, it means that your weblog does work... (sent by Anonymous on December 1, 2006 1:17 PM about Welcome to the symfony weblog!)		
Thank you for your feedback. It really helps to see people understanding you. (sent by Myself on December 1, 2006 1:17 PM about Welcome to the symfony weblog!)		
Mine is not bad either. I'd like to see more deers, though. (sent by John Doe on December 1, 2006 1:17 PM about Life is beautiful)		
How can you be so positive? There are so many subjects to worry about out there! (sent by Anonymous on December 1, 2006 1:17 PM about Life is beautiful)		
Why is it always like that, people quarrelling as soon as they have room to express themselves? (sent by Myself on December 1, 2006 1:17 PM about Life is beautiful)		
5 results		



请注意 tabular 布局需要一个由字段名组成的数组作为 display 键的值，但是 stacked 布局使用 params 键为每个记录生成 HTML。不过，在 stacked 布局中也需要使用 display 数组，决定需要哪些列作为排序的表头。

过滤结果

在 list 视图里可以加一些过滤器。使用过滤器，可以减少显示的结果并且更快的找到需要的记录。过滤器在 filters 键下设置，它的值是字段名组成的数组。例如，在 comment 模块的 list 视图增加 author_id, author 和 created_at 字段的过滤器，如例 14-23，会显示一个类似图 14-16 所示的过滤框。要让这段配置正常工作需要在 Article 类增加一个 __toString() 方法（例如，返回文章标题）。


例 14-23 - 在 list 视图里设置过滤器，位于 modules/comment/config/generator.yml

```
list:
  filters: [article_id, author, created_at]
  layout: stacked
  params: |
    %=content%% <br />
    (sent by %%author%% on %%created_at%% about %%article_link%%)
  display: [created_at, author, content]
```


图 14-16 - comment 模块的 list 视图里的过滤器

comment list

Published on	Author	Body
Well, if this comment displays, it means that your weblog does work... (sent by Anonymous on December 1, 2006 1:17 PM about Welcome to the symfony weblog!)		
Thank you for your feedback. It really helps to see people understanding you. (sent by Myself on December 1, 2006 1:17 PM about Welcome to the symfony weblog!)		
Mine is not bad either. I'd like to see more deers, though. (sent by John Doe on December 1, 2006 1:17 PM about Life is beautiful)		
How can you be so positive? There are so many subjects to worry about out there! (sent by Anonymous on December 1, 2006 1:17 PM about Life is beautiful)		
Why is it always like that, people quarrelling as soon as they have room to express themselves? (sent by Myself on December 1, 2006 1:17 PM about Life is beautiful)		
5 results		




filters

Article: 

Author:

Published on:

 reset

symfony 显示的过滤器取决于字段类型：

- 对于文字字段（比如 comment 模块的 author 字段），过滤器会是一个可以使用通配符（*）的文字输入框。
- 对于外键字段（比如 comment 模块里的 article_id 字段），过滤器会是一个显示相关表记录的下拉列表。至于普通的 object_select_tag()，下拉列表的选项是由相关类的 __toString() 方法的结果组成的。
- 对于日期字段（比如 comment 模块的 created_at 字段），过滤器会是一对可以让人选择时间范围的日期控件。
- 对于布尔字段，过滤器会是一个包含 true、false 和 true or false 选项的下拉列表——最后一个值会重设过滤器。

正如你可以在列表中使用局部模板，你也可以在过滤器中使用局部模板来实现一个 symfony 处理不了的过滤器。例如，假设 state 字段只可能有两个取值（open 与 closed），但是由于某些原因你直接在字段里存放这些值而不是使用表关联。symfony 会自动给这个字段(string 类型)一个文字搜索，不过你想要的可能是下拉列表。这用局部模板很容易实现。例 14-24 是实现的方法。

例 14-24 - 使用局部模板过滤器

```
// 定义局部模板，位于 templates/_state.php
<?php echo select_tag('filters[state]', options_for_select(array(
    '' => '',
    'open' => 'open',
    'closed' => 'closed',
), isset($filters['state']) ? $filters['state'] : '')) ?>
```

```
// 在过滤器列表里增加局部模板过滤器，位于 config/generator.yml
list:
  filters:      [date, _state]
```

注意这里用到的局部模板可以访问到`$filters` 这个变量，这对取得过滤器当前值很有用。

还有一个选项对搜索空值很有用。假设你想找所有没有作者的留言，问题是如果你不填写作者过滤器的输入框，`symfony` 会忽略它。解决办法是把字段的 `filter_is_empty` 属性设置成 `true`，如例 14-25 所示，这样会多显示一个复选框让你可以寻找空值如图 14-17。

例 14-25 - 给 `list` 视图的 `author` 过滤器增加空值选项

```
list:
  fields:
    author:  { filter_is_empty: true }
  filters:  [article_id, author, created_at]
```

图 14-17 - 允许过滤空的 `author` 的值



The image shows a web interface for filtering search results. It is titled "filters" in a grey header. Below the header, there are three filter sections. The first section is labeled "Article:" and contains a dropdown menu with a downward arrow. The second section is labeled "Author:" and contains a text input field and a checkbox labeled "is empty". The third section is labeled "Published on:" and contains a date range selector with two input fields and a calendar icon. At the bottom of the filter section, there are two buttons: "reset" and "filter".

列表排序

在 `list` 视图里，表头是可以改变列表排列顺序的超链接，如图 14-18 所示。这些表头在 `tabular` 和 `stacked` 布局中都会显示。点击这些链接会用 `sort` 参数重新载入页面从而改变列表显示顺序。

图 14-18 - `list` 视图的表头可以控制排列顺序

List of Articles

Id	Title	nb comments	Created at
1	Welcome to the symfony weblog!	2	December 1, 2006 1:17 PM
2	Life is beautiful	3	December 1, 2006 1:17 PM
2 results			



可以重用这里的参数排列来指向一个直接按照某个字段排列的列表：

```
<?php echo link_to('Comment list by date', 'comment/list?
sort=created_at&type=desc' ) ?>
```

也可以在 generator.yml 里为 list 视图设置一个默认的 sort 排列顺序。格式如例 14-26 所示。

例 14-26 - 给 list 视图一个默认的排列顺序

```
list:
  sort:  created_at
  # 另一种格式，指定一个特定的排列方向
  sort:  [created_at, desc]
```

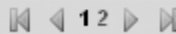

NOTE 只有实际的字段可以转化成排序链接——自定义字段或者局部模板字段不可以。

自定义分页

自动生成的管理界面可以有效的处理大规模的表，因为 list 自动进行分页。当表的记录数大于每页最大记录数的时候，翻页控件会出现在列表的底部。例如，图 14-19 显示的是一个有 6 条记录的留言表，但是每页只显示 5 条记录。分页能提高显示速度，因为只有需要显示的记录才会从数据库里读取，它还提高了易用性，因为即使是有上百万条记录的表都可以用生成的管理界面进行管理。

图 14-19 - 分页控件会显示在长的列表里

comment list

Published on	Author	Body
Well, if this comment displays, it means that your weblog does work... (sent by Anonymous on December 1, 2006 1:17 PM about Welcome to the symfony weblog!)		
Thank you for your feedback. It really helps to see people understanding you. (sent by Myself on December 1, 2006 1:17 PM about Welcome to the symfony weblog!)		
Mine is not bad either. I'd like to see more deers, though. (sent by John Doe on December 1, 2006 1:17 PM about Life is beautiful)		
How can you be so positive? There are so many subjects to worry about out there! (sent by Anonymous on December 1, 2006 1:17 PM about Life is beautiful)		
Why is it always like that, people quarrelling as soon as they have room to express themselves? (sent by Myself on December 1, 2006 1:17 PM about Life is beautiful)		
6 results		
		

可以通过指定 `max_per_page` 参数来自定义每页显示的记录条数：

```
list:
    max_per_page: 5
```

使用 join 加快页面速度

默认情况，管理界面生成器使用简单的 `doSelect()` 方法来获取一个列表。不过，如果在列表里面使用了关联对象，数据库查询次数就会飞速增长。例如，如果你想在留言列表里显示文章的名字，每个留言需要多做一次查询来取得相关的 `Article` 对象。所以需要强制分页系统使用 `doSelectJoinXXX()` 方法来优化查询次数。这可以通过 `peer_method` 来设置。

```
list:
    peer_method: doSelectJoinArticle
```

第 18 章详细解释了 `join` 的概念。

edit 视图相关定制

在 `edit` 视图里，用户可以修改指定记录的每一个字段的值。symfony 根据字段的数据类型确定表单控件的类型，然后生成一个 `object*_tag()` 辅助函数，并将当前对象与控件属性传给这个辅助函数。例如，如果文章 `edit` 视图配置里设置了用户可以编辑 `title` 字段：

```
edit:
    display: [title, ...]
```

那么由于 `schema` 里字段的类型是 `varchar`，`edit` 页面会显示一个用来修改

title 的普通的文字输入框。

```
<?php echo object_input_tag($article, 'getTitle') ?>
```

修改表单控件类型

默认的数据类型到表单控件的转换规则如下：

- integer, float, char, varchar(size) 类型的字段会在 edit 视图里以 object_input_tag() 的形式出现。
- longvarchar 类型的字段会以 object_textarea_tag() 的形式出现。
- 外键字段会以 object_select_tag() 的形式出现。
- boolean 类型的字段会以 object_checkbox_tag() 的形式出现。
- timestamp 与 date 类型的字段会以 object_input_date_tag() 的形式出现。

你可能会想重写这些规则为某个字段指定一个特殊的表单控件类型。要做到这点，需要设置 fields 定义下的字段 type 参数，指定一个表单辅助函数的名字。生成的 object_*_tag() 的属性，可以通过 params 参数来修改。如例 14-27 所示。

例 14-27 - 在 edit 视图里设置一个自定的表单控件类型与参数

```
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:   Comment
    theme:         default

  edit:
    fields:
      ## 不显示表单控件，只显示文本
      id:          { type: plain }
      ## 表单控件不可编辑
      author:      { params: disabled=true }
      ## 文本编辑框(object_textarea_tag)
      content:     { type: textarea_tag, params: rich=true
css=user.css tinymce_options=width:330 }
      ## 下拉列表(object_select_tag)
      article_id:  { params: include_custom=Choose an article }
      ...
```

params 参数会作为选项传给 object_*_tag()。例如，之前的 article_id 的 params 定义会产生如下的模板：

```
<?php echo object_select_tag($comment, 'getArticleId',  
'related_class=Article', 'include_custom=Choose an article') ?>
```

这就是说 edit 视图里的表单辅助函数的所有参数都可以通过配置文件定制。

处理局部模板字段

局部模板不仅可以在 list 视图里使用，也可以在 edit 视图里使用。不同之处是在 edit 视图里你需要亲自处理局部模板，自己写动作(action)里的根据局部模板传来的数据更新字段的部分代码。symfony 知道如何处理普通的字段（实际的字段），它也会自己猜测的出如何处理你的局部模板里传来的数据。

例如，假设一个 User 类的管理界面模块有 id, nickname 和 password 三个字段。网站管理员需要能根据请求修改用户的密码，但是处于安全考虑 edit 视图不能显示密码的值。表单里应该显示一个空的密码输入框。例 14-28 里的配置可以实现上面的需求。

例 14-28 - 在 edit 视图里包含一个局部模板字段

```
edit:  
  display:      [id, nickname, _newpassword]  
  fields:  
    newpassword: { name: Password, help: Enter a password to  
change it, leave the field blank to keep the current one }
```

templates/_newpassword.php 局部模板文件的内容如下：

```
<?php echo input_password_tag('newpassword', '') ?>
```

注意这个局部模板使用的是简单的表单辅助方法，而不是对象表单辅助方法，因为这里不需要从当前 User 独享取得密码的值来生成这个表单控件——这可能泄漏用户的密码。

现在，要在动作(action)里使用表单的值来更新对象，你还需要扩展动作里的 updateUserFromRequest() 方法。这需要在动作文件里增加一个包含处理这个模板字段的自定义行为的同名方法，如例 14-29。

例 14-29 - 在动作里处理局部模板， 位于
modules/user/actions/actions.class.php

```
class userActions extends sfActions  
{  
  protected function updateUserFromRequest()  
}
```

```

{
    // 处理局部模板字段的输入
    $password = $this->getRequestParameter('newpassword');

    if ($password)
    {
        $this->user->setPassword($password);
    }

    // 让 symfony 处理其他的字段

    parent::updateUserFromRequest();
}
}

```

NOTE 在实际的应用中，user/edit 视图通常应该包含两个密码字段，第二个用来跟第一个作比对从而避免输入错误。你可以在第 10 章找到，这可以通过 validator 来实现。自动生成的管理界面模块与普通模块一样也可以从这种机制中获益。

处理外键

如果你的 schema 里定义了表间关系，那么自动生成的管理界面模块会利用定义表间关系并能自动处理它们，这极大的简化了关系管理。

一对多关系

管理界面生成器可以很好的处理 1-n(一对多)的表关系。在图 14-1 里，表 blog_comment 与 blog_article 表通过 article_id 字段相关联。如果你用管理界面生成器初始化 Comment 类的模块，comment/edit 动作会自动用下拉列表来显示 blog_article 表里的记录作为 article_id 字段的表单控件（请看图 14-9）。

另外，如果给 Article 对象定义一个 __toString() 方法，下拉列表的选项文字会从原来的主键变成这个方法的返回值。

如果你要在 article 模块里显示一篇文章的留言列表（n-1 关系），你需要用一个局部模板字段来对这个模块进行定制。

多对多关系

symfony 也可以处理多对多关系，但是在 schema 里没有办法定义它们，需要在 generator.yml 里面通过一些额外的参数来实现。

多对多关系的实现需要一个中间表。例如，如果 `blog_article` 与 `blog_author` 之间存在 `n-n` 关系（一篇文章可以有多个作者，很明显，一个作者可以写多篇文章），你的数据库肯定会有一个 `blog_article_author` 或者类似的表，如图 14-20。

图 14-20 - 使用 “`through_class` 中间类” 实现多对多关系



这个模型有一个叫 `ArticleAuthor` 的类，管理界面生成器只需要它就可以了，不过你还要把字段的 `through_class` 参数设置成这个类。

例如，在基于 `Article` 类生成的模块里，可以通过例 14-30 里的 `generator.yml` 文件新增一个代表 `Author` 类的 `n-n` 关联的字段。

例 14-30 - 通过 `through_class` 参数处理多对多关系

```
edit:
  fields:
    article_author: { type: admin_double_list, params:
through_class=ArticleAuthor }
```

这个字段会处理存在的对象之间的关联，所以下拉列表就显得不够了。需要给它设置一个特殊的表单控件。symfony 提供了三个表单控件来处理多对多关系（如图 14-21）：

- `admin_double_list` 是两个展开的选择列表组成的选择控件，它还包含在第一个列表（可选元素）和第二个列表（选中元素）切换元素的按钮。
- `admin_select_list` 是一个可以用来选择多个元素的展开选择控件。
- `admin_check_list` 是由一组复选框组成的。

图 14-21 - 可用于多对多关系的控件



增加交互

管理界面模块允许用户进行常见的 CRUD 操作，不过你也可以增加你自己的交互或者对某个视图的交互做限制。例如，例 14-31 里的交互定义包括了所有 article 模块的默认 CRUD 动作。

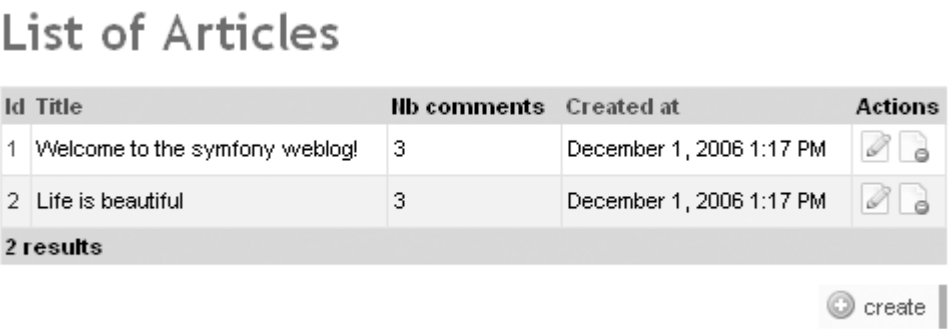
例 14-31 - 给每个视图定义交互，位于
backend/modules/article/config/generator.yml

```
list:
  title:          List of Articles
  object_actions:
    _edit:        ~
    _delete:      ~
  actions:
    _create:      ~

edit:
  title:          Body of article %%title%%
  actions:
    _list:        ~
    _save:        ~
    _save_and_add: ~
    _delete:      ~
```

在 list 视图里有两组动作设置：针对每个对象的动作和针对整个页面的动作。例 14-31 里定义的交互结果如图 14-22 所示。每行都有一个编辑记录的按钮还有一个删除按钮。在列表的底部，有一个增加记录的按钮。

图 14-22 - list 视图的交互



在 edit 视图里，由于一次只修改一条记录，所以只要定义一组动作。例 14-31 里定义的 edit 视图的交互结果如图 14-23 所示。save 还有 save_and_add 动作都会把当前对象写回数据库，不同的是 save 动作保存了记录之后会回到当前记录的 edit 视图，而 save_and_add 动作会显示一个空的 edit 视图来新增一条记录。save_and_add 在一次增加多条记录的时候很有用。delete 按钮的位置与其他的按钮分开，这样用户不容易点错。


交互的名字以下划线(_)开头告诉 symfony 使用这个动作默认的图标与动作。管理界面生成器可以理解的交互有 `_edit`、`_delete`、`_create`、`_list`、`_save`、`_save_and_add` 和 `_create`。


图 14-23 - edit 视图的交互


Body of article Life is beautiful


Content:

The purpose of a weblog is usually to talk about one's mood. Mine is great today. How is yours?

 list

 save

 save and add

 delete

不过你也可以指定一个自定义的交互，这样需要指定一个不以下划线开头的名字，如例 14-32。


例 14-32 - 定义一个自定义交互

```
list:
  title:          List of Articles
  object_actions:
    _edit:        -
    _delete:      -
    addcomment:   { name: Add a comment, action: addComment,
icon: backend/addcomment.png }
```

如图 14-24，列表里面的每篇文章现在有一个显示 `addcomment.png` 的按钮。点击这个按钮触发当前模块的 `addComment` 动作。当前对象的主键会自动附加到请求参数里。

图 14-24 - list 视图的自定义交互

List of Articles

Id	Title	lib comments	Created at	Actions
1	Welcome to the symfony weblog!	3	December 1, 2006 1:17 PM	  
2	Life is beautiful	3	December 1, 2006 1:17 PM	  
2 results				

 create

Add a comment

addComment 的代码如例 14-33 所示。

例 14-33 - 自定义交互的动作代码, actions/actions.class.php

```
public function executeAddComment()
{
    $comment = new Comment();
    $comment->setArticleId($this->getRequestParameter('id'));
    $comment->save();

    $this->redirect('comment/edit?id='.$comment->getId());
}
```

关于交互最后说一下：如果想要完全抑制某种交互，请像例 14-34 里那样使用空列表。

例 14-34 - 去掉 list 视图里的所有动作

```
list:
  title:      List of Articles
  actions:    {}
```

表单验证

如果你看一下项目 cache/目录里生成的 _edit_form.php 模板，你会发现表单的字段使用了一个特殊的命名规则。在生成的 edit 视图里，控件的名字由模块名与方括号括起来的字段名组成。

例如，article 模块的 edit 视图有一个 title 字段，对应的模板可能会像例 14-35 中的那样，模板里的控件名将会是 article[title]。

例 14-35 - 生成的控件名格式

```
// generator.yml
generator:
  class:      sfPropelAdminGenerator
  param:
    model_class:  Article
    theme:        default
    edit:
      display: [title]

// 生成的 _edit_form.php 模板
<?php echo object_input_tag($article, 'getTitle',
```



```
array('control_name' => 'article[title]')) ?>
```

```
// 返回的 HTML 代码
```

```
<input type="text" name="article[title]" id="article[title]"
value="My Title" />
```

这对内部表单处理过程有很多好处。不过，根据第 10 章里介绍的内容，这会使表单验证有点麻烦，你要把 `fields` 定义里的方括号, [] 改成大括号 {}。并且，如果要在 `validator` 的参数里使用字段名，需要用它在 HTML 里面的代码（也就是方括号，不过要用引号括起来）。生成表单的特殊的 `validator` 语法的详情请参考例 14-36。

例 14-36 - 管理界面生成表单的 `validator` 文件语法

```
## 用大括号替换方括号
```

```
fields:
  article{title}:
    required:
      msg: You must provide a title
    ## validator 的参数里，使用原始的字段名并用引号引用
    sfCompareValidator:
      check: "user[newpassword]"
      compare_error: The password confirmation does not match the
password.
```

使用证书限制用户动作

对于某个管理界面模块，可用字段和交互可以根据当前登录用户的证书而变化（详情见第 6 章 `symfony` 的安全功能部分）。

生成器里的字段可以使用 `credentials` 来限定只有拥有特定证书的用户使用。这在 `list` 视图和 `edit` 视图里都会起作用。另外，生成器可以根据证书隐藏交互。例 14-37 演示了这些功能。

例 14-37 - 在 `generator.yml` 里使用证书

```
## 只有拥有 admin 证书的用户可以看到 id 字段
```

```
list:
  title: List of Articles
  layout: tabular
  display: [id, =title, content, nb_comments]
  fields:
    id: { credentials: [admin] }
```

```
## 只有拥有 admin 证书的用户可以使用 addcomment 交互
```

```
list:
  title:          List of Articles
  object_actions:
    _edit:        -
    _delete:      -
    addcomment:   { credentials: [admin], name: Add a comment,
action: addComment, icon: backend/addcomment.png }
```

修改生成模块的外观

你可以修改生成模块的外观来配合已经存在的设计，不仅可以使使用你自己的样式表，还可以重写默认模板。

使用自定义样式表

由于生成的 HTML 内容是结构化的，所以你可以尽情的修改它的外观。

你可以为管理模块定义自己的 CSS 取代默认的 CSS，只要在 `generator.yml` 里加一个 `css` 参数就可以了，如例 14-38。

例 14-38 - 使用自定义样式表取代默认样式表

```
generator:
  class:          sfPropelAdminGenerator
  param:
    model_class:   Comment
    theme:         default
    css:           mystylesheet
```

另外，也可以使用 `view.yml` 为每个视图重写样式设置。

增加自定义头部与尾部

`list` 视图和 `edit` 视图会自动载入一个头部和尾部局部模板。默认情况管理界面模块的 `templates/` 目录里没有这个局部模板，不过只要按照下面的名字自己建立这些模板，它们就会被自动载入：

```
_list_header.php
_list_footer.php
_edit_header.php
_edit_footer.php
```

例如，如果你想在 `article/edit` 视图增加一个自定义头部，按照例 14-39 的内容建立一个名叫 `_edit_header.php` 的文件。不需要其他配置它就会起作用。

例 14-39 - edit 头部局部模板，位于
modules/articles/template/_edit_header.php

```
<?php if ($article->getNbComments() > 0): ?>
    <h2>This article has <?php echo $article->getNbComments() ?>
comments.</h2>
<?php endif; ?>
```

注意 edit 视图里的局部模板总可以通过与模块名相同的变量访问到当前对象，另外 list 视图里的局部模板总可以通过 \$pager 变量访问到当前页的信息。

SIDEBAR 使用自定参数调用管理界面的动作

管理界面模块可以通过 link_to() 辅助函数里的 query_string 参数来接受自定参数。例如，给前面的 `_edit_header` 局部模板增加一个给文章留言的链接，要这样写：

```
getNbComments() > 0): ?>

This article has getNbComments().' comments', 'comment/list',
array('query_string' => 'filter=filter&filters%5Barticle_id
%5D='.$article->getId())) ?>
```

这个 query_string 是编码过的版本，这样更可靠

```
'filter=filter&filters[article_id]='.$article->getId()
```

它只过滤显示跟 \$article 有关的留言。使用 query_string 参数，你可以指定排列顺序和过滤器来显示一个特殊的列表视图。这对自定义交互也很有用。

自定义主题

还有一些继承自框架的局部模板可以在 templates/ 目录里被重写来满足你的特殊需求。

生成的模板被分成小块，它们可以单独重写，动作也可以单独修改。

不过，如果你想用同样的方式在多个模块里多次重写，你应该作一个可重用的主题。主题是一系列用于管理界面生成器的完整的模板与动作，要使用一个主题需要在 generator.yml 文件的开头指定这个主题。默认的 symfony 主题的文件放在 \$sf_symfony_data_dir/generator/sfPropelAdmin/default/ 目录。

这些主题文件需要放在项目目录的 data/generator/sfPropelAdmin/[theme_name]/template/目录下，可以通过复制默认主题的文件(在 \$sf_symfony_data_dir/generator/sfPropelAdmin/default/template/目录)快速地建立一个新主题。用这种方法，你要确定你的自定义主题里面包含下面这些必须的文件：

```
// 局部模板，[theme_name]/template/templates/
_edit_actions.php
_edit_footer.php
_edit_form.php
_edit_header.php
_edit_messages.php
_filters.php
_list.php
_list_actions.php
_list_footer.php
_list_header.php
_list_messages.php
_list_td_actions.php
_list_td_stacked.php
_list_td_tabular.php
_list_th_stacked.php
_list_th_tabular.php

// 动作，[theme_name]/template/actions/actions.class.php
processFilters() // 处理请求的过滤器
addFiltersCriteria() // 把过滤器增加到条件对象
processSort()
addSortCriteria()
```

注意这些模板文件实际上是模板的模板，也就是说，这个 PHP 文件会被用一个特殊的工具解析并且根据生成器设置生成模板（这被称作编译阶段）。生成的模板也许要包含实际浏览的时候执行的 PHP 代码，所以模板的模板使用了特殊的语法来使 PHP 代码在第一阶段不被执行。例 14-40 是一个默认模板的模板的实际内容。

例 14-40 - 模板的模板的语法

```
<?php foreach ($this->getPrimaryKey() as $pk): ?>
[?php echo object_input_hidden_tag($<?php echo $this-
>getSingularName() ?>,'get<?php echo $pk->getPhpName() ?>') ?]
<?php endforeach; ?>
```

在这个例子里，<?里的 PHP 代码会立即执行（编译时），[?里的只在执行时执

行，但模板引擎最终会把[?标签转换成<?标签，所以最后的模板看上去是这样的：

```
<?php echo object_input_hidden_tag($article, 'getId') ?>
```

处理模板的模板需要特别小心，所以要作一个新的主题最好还是从默认模板开始，一步一步修改他，多作测试。

TIP 你也可以把生成器的模板打包做成 plug-in，这会使它更容易在多个项目中重用。详情请参考第 17 章。

—

SIDEBAR 建立你自己的生成器

脚手架与管理界面生成器都使用了一系列 symfony 内部的自动地在缓存里生成代码的组件，还有主题和模板的模板。

这意味着 symfony 提供了所有的建立你自己的生成器所需要的工具，你自己的生成器可能会与现有的生成器类似或者完全不同。模块的生成是由 sfGeneratorManager 类的 generate() 方法来管理的。例如，要生成一个管理界面，symfony 内部会调用下面的代码：

```
$generator_manager = new sfGeneratorManager();  
$data = $generator_manager->generate('sfPropelAdminGenerator',  
$parameters);
```

如果你想作自己的生成器，你需要看 sfGeneratorManager 和 sfGenerator 的 API 文档，还要参考 sfAdminGenerator 和 sfCRUDGenerator 类。

总结

要快速建立模块或者自动生成后台，基础是定义良好的数据库设计（schema）和对象模型。你可以修改脚手架的 PHP 代码，但是管理界面模块的大部分修改是通过配置文件进行的。

generator.yml 文件是后台编程的中心。它可以完全定制 list 视图和 edit 视图的内容、功能以及外观。字段的标签、提示、过滤器、排序还有页面记录数、控件类型、外键关系、自定义交互及其证书都可以通过直接修改 YAML 文件来实现，而不用写 PHP 代码。

如果管理界面生成器不支持你需要的功能，局部模板字段还有重写动作为你提供了完整的扩展性。另外你可以使用管理界面生成器的主题机制来重用你的修

改。