第 5 章　配置 symfony

为了达到简单易用的目的，symfony 定义了一些惯例，这些惯例能够满足大多数情况的需求。另一方面，使用一系列简单而强大的配置文件，我们可以定制这个框架及应用程序的几乎所有的地方。使用这些配置文件可以为程序增加一些特殊的参数。

这一章介绍配置系统如何工作：

- symfony 配置信息保存在 YAML 格式的文件里，当然也可以换成其它格式。
- 配置文件分成项目、应用程序、模块这几个等级，分别存放在项目目录中对应子目录里。
- 你可以定义几套不同的配置文件，symfony 里面的一套配置文件称之为环境。
- 配置文件里面定义的值可以在 PHP 代码里面取得。
- 另外，symfony 可以识别 YAML 里面的 PHP 代码等，这使得配置系统更灵活。

配置系统

不论什么用途，大多数 web 应用程序都有一些共同的特征。例如，有些区域只允许一部分用户访问；很多页面共用一个布局；表单填写验证失败后自动把用户输入的内容放进表单。框架定义了一些实现这些特性的结构，开发者通过配置系统进一步的调整它们。这种策略可以节省大量开发时间，因为很多改变并不用改写代码，尽管实现这些改变需要很多代码。这种策略也更有效率，因为这些信息可以存放在容易识别的位置。

但是，这样的做法有两个严重缺点：

- 开发者最后整天不停的写复杂的 XML 文件。
- 使用 PHP 处理每个请求花费的时间更多了。

考虑到这些缺点，同时 symfony 又要充分利用配置文件的优点。事实上，symfony 的配置系统的目标是：

- 强大：所有的可以配置的东西都可以用配置文件配置
- 简单：很多配置信息不会出现在普通的应用程序里，因为它们很少需要改变
- 容易：开发者可以很容易的阅读，建立，修改配置文件
- 可定制：默认的配置语言是 YAML，但也可以换成 INI、XML，或者是别的格式

- 快速：应用程序本身不用处理配置文件，由配置系统处理配置文件，把配置文件编译成能够快速执行的 PHP 代码

YAML 语法与 symfony 惯例

symfony 默认使用 YAML 格式存放配置信息，而不用传统的 INI 或者 XML 格式。YAML 通过缩进表示结构而且写起来很快。它的特点与基本规则在第一章里面已经讲到了。不过，在写 YAML 的时候，你还需要记住几条规则。本节将介绍最常用的几个规则。想完整的了解 YAML，请访问 YAML 网站(http://www.yaml.org/)。

首先，绝不要在 YAML 文件里使用制表符(tab)，应该使用空格。YAML 解析器不能解析制表符，所以请使用空格来缩进(在 symfony 里面使用两个空格缩进)，如例 5-1。

例 5-1 - YAML 文件禁用制表符(tab)

```
# 绝不用制表符
all:
-> mail:
-> -> webmaster:  webmaster@example.com

# 应该使用空格
all:
  mail:
    webmaster: webmaster@example.com
```

如果你的参数是以空格开始或者结束的字符串，应使用单引号把它包起来。如果一个字符串参数包含特殊字符，也要用单引号包起来，如例 5-2。

例 5-2 - 非标准的字符串要用单引号包起来

```
错误 1: This field is compulsory
错误 2: '  This field is compulsory  '
错误 3: 'Don''t leave this field blank'    # 必须用两个单引号来表示字符串中的'
```

利用特殊字符头(> 或 |)与一个缩进，长的字符串可以跨行表示。如例 5-3。

例 5-3 - 定义长的多行字符串

```
accomplishment: >           # 由>开头的折叠式
  Mark set a major league   # 每一个换行被折叠成一个空格
  home run record in 1998.  # 使得 YAML 可读性更强
stats: |                    # 由|开头的原始式
```

```
  65 Home Runs                # 所有的换行都被保留
  0.278 Batting Average       # 缩进不会在结果里出现
```

如果要定义数组，需要用方括号把元素括起来或者使用展开的减号语法，如例5-4。

例 5-4 - YAML的数组语法

```
# 数组语法的简写
players: [ Mark McGwire, Sammy Sosa, Ken Griffey ]

# 数组的展开语法
players:
  - Mark McGwire
  - Sammy Sosa
  - Ken Griffey
```

如果要定义关联数组或者说哈希表，要用大括号把元素括起来，键与值key: value中间保留一个空格。也可以用展开语法，每一个新的键增加一个缩进与换行，如例5-5。

例 5-5 - YAML关联数组语法

```
# 错误的语法，冒号后缺少空格
mail: {webmaster:webmaster@example.com,contact:contact@example.com}

# 关联数组的正确简写
mail: { webmaster: webmaster@example.com, contact:
contact@example.com }

# 关联数组的展开语法
mail:
  webmaster: webmaster@example.com
  contact:   contact@example.com
```

定义一个布尔值时，on、1或者true代表肯定值，off、0、或者false代表否定值。如例5-6。

例 5-6 - YAML 布尔值语法

```
true_values:   [ on, 1, true ]
false_values:  [ off, 0, false ]
```

请不要吝啬使用注释(以井号#开头)还有空格，这会使注释文件更易读，如例5-7。

例 5-7 - YAML 注释语法与值对齐

```
# 这是一个注释
mail:
  webmaster: webmaster@example.com
  contact:   contact@example.com
  admin:     admin@example.com   # 多的空格可以帮助对齐
```

在某些 symfony 配置文件里面，你会发现一些行以#开头(YAML 解析器会忽略这些行),但这些行看上去像普通的设置行。这是一个 symfony 的惯例：默认设置，从 symfony 内核里其他的 YAML 文件里面继承的设置，这些会在你的应用程序配置文件里面出现并用#注释起来，给你参考。如果你想改变这些参数，只要把注释去掉就可以了。如例 5-8。

例 5-8 - 注释里的默认配置

```
# 缓存的默认值是关闭
settings:
# cache: off

# 如果你想修改这个值，去掉注释
settings:
  cache: on
```

symfony 有时会把一些参数定义分类。一个分类里面的所有设置都放在分类头下面。把长的 key: value 列表分组能增强可读性。分类头以点(.)开头。如例 5-9。

例 5-9 - 分类头与键类似，但是以.开头

```
all:
  .general:
    tax:        19.6

  mail:
    webmaster:  webmaster@example.com
```

在这个例子里，mail 是一个键，general 只是一个分类头。分类头可以当作不存在，如例 5-10。tax 参数实际上是 all 键的直接子元素。

例 5-10 - 分类头只用于增强可读性，实际上可以忽略

```
all:
  tax:          19.6

  mail:
```

```
    webmaster:  webmaster@example.com
```

SIDEBAR 如果你不喜欢 YAML

YAML 只是一个给 PHP 代码定义设置的界面，所以 YAML 里面定义的配置信息都会被转换成 PHP 代码。浏览一个应用程序，查看他的缓存的配置信息(例，在 cache/myapp/dev/config/)。你会发现 YAML 配置对应的 PHP 文件。这一章后面我们会详细介绍配置缓存。

好消息是如果你不喜欢 YAML 文件，你可以自己动手，使用 PHP 代码或者其他的格式(XML,INI 等)。在本书中，你会遇到其他的不使用 YAML 定义配置的方法，在第 19 章你会了解如何替换 symfony 的配置文件处理器。如果你用好它们，你可以利用这些技巧绕开配置文件或者定义你自己的格式。

救命，YAML 文件把我的程序搞死了

YAML 文件会被解析成 PHP 哈希与数组，然后这些值会在程序的不同地方改变视图，控制器或者模型的行为。很多时候，配置文件里面的问题直到用到的时候才被察觉。而且，显示的错误信息通常不是明显与 YAML 配置文件相关的。

如果改变了配置文件之后程序突然停止运行了，应该检查一下你是否犯了下面的常见的 YAML 错误：

- 键和值中间缺少空格：

```
key1:value1       # 冒号:后缺少空格
```

- 数组里面的键应该按照同样的方式缩进：

```
all:
  key1:  value1
   key2: value2  # 缩进与其他的数组元素不同
  key3:  value3
```

- 键或值里有 YAML 保留字符而且没有用单引号：

```
message: tell him: go way    # :, [, ], { and } 是 YAML 保留字符
message: 'tell him: go way'  # 正确的语法
```

- 修改一个被注释了的行：

```
# key: value      # 由于前面的#,这行永远不会生效
```

- 同一级别相同键的值设置了两次：

```
key1: value1
key2: value2
key1: value3      # key1 定义了两次，取最后一次定义的值
```

- 你认为设置值应该是一个特定的类型，实际上如果你不转换它，设置值永远是字符串：

```
income: 12,345    # 除非你转换它，否则它永远是字符串
```

配置文件概述

配置信息按照目的存放在不同的文件里。这些文件包含参数定义或者设置。一些参数可以在不同的级别覆盖(项目，应用程序，模块)；一些只针对特定级别。本节将介绍这些配置文件，第 19 章将更深入的介绍配置文件。

项目配置

symfony 项目有一些默认配置文件。下面是 myproject/config/目录下面的配置文件：

- config.php：这是所有页面或者命令行方式 PHP 脚本执行的第一个文件。它包含 symfony 框架的路径，你可以把这个路径指定到另外一个 symfony 框架。如果你在这里增加一些 define 语句，这样定义的常量在项目的每个应用程序都可以访问得到。第 19 章将会深入介绍这个文件的使用。
- databases.yml：这个文件用来存放数据库连接设置(主机，登录名，密码等)。第 8 章有更详细的介绍。这个文件的配置信息可以在应用程序这一级别覆盖。
- properties.ini：这个文件存放命令行工具需要的参数，包括项目名称，远程服务器的连接设置。第 16 章将详细介绍此文件的功能。
- rsync_exclude.txt：这个文件定义同步服务器的时候哪些文件不需要同步。详见第 16 章。
- schema.yml 与 propel.ini：这两个文件是 Propel(symfony 的 ORM 层)的数据访问配置文件。它们用来使 Propel 与 symfony 的类还有项目的数据协同工作。propel.ini 是自动生成的，所以你不用修改它。如果你不用 Propel，就不需要这些文件。这两个文件的使用详见第 8 章。

这些文件多数时候是被外部组件或者命令行使用，或者 symfony 在 YAML 解析程序载入之前就要用到它们。所以有些文件不是 YAML 格式。

应用程序配置

配置的主要部分是应用程序配置。前端控制器(在 web/目录)里定义主要的常量，YAML 文件在应用程序目录的 config/目录里，i18n/目录里是国际化需要的文件，还有一些在框架文件里隐藏着的但是很有用的其他项目配置信息。

前端控制器配置

前端控制器里存放了应用程序最开始的配置信息；它是一个请求最开始执行的脚本。请看例 5-11 中默认的 web/index.php。

例 5-11 - 默认的生产环境前端控制器

```
<?php

define('SF_ROOT_DIR',    dirname(__FILE__).'/..');
define('SF_APP',         'myapp');
define('SF_ENVIRONMENT', 'prod');
define('SF_DEBUG',       true);

require_once(SF_ROOT_DIR.DIRECTORY_SEPARATOR.'apps'.DIRECTORY_SEPARATOR.SF_APP.DIRECTORY_SEPARATOR.'config'.DIRECTORY_SEPARATOR.'config.php');

sfContext::getInstance()->getController()->dispatch();
```

定义了应用程序名(myapp)与环境(prod)之后，先载入通用配置文件，然后继续分派请求。这里定义了一些有用的常量：

- SF_ROOT_DIR：项目根目录 (一般情况请保留默认值，除非想变更目录结构)。
- SF_APP：项目中的应用程序名。需要它来生成文件路径。
- SF_ENVIRONMENT：环境名 (prod, dev, 或者其他你定义的本项目的环境)。用来决定使用哪一套配置信息。本章稍后会解释环境的概念。
- SF_DEBUG：是否启用调试模式 (详见第 16 章)。

如果你要改变这些值，那么你可能需要另一个前端控制器。下一章将介绍前端控制器以及如何新建一个前端控制器。

SIDEBAR 根目录可以在任何地方

只有 web 根目录(symfony 项目的 web/目录)里的脚本对外界公开。前端控制器脚本，图片，样式表，还有 JavaScript 文件是公开的。其他文件必须放在服务器 web 根目录之外--也就是说其他任何地方。

前端控制器通过SF_ROOT_DIR这个路径访问项目的非公开的文件。一般来说，项目根目录就是web/目录的上一层目录。但是你可以选择一个完全不同的文件结构。假设你的主目录结构由两个目录组成，一个公开另外一个私有：

```
symfony/     # 私有区域
  apps/
  batch/
  cache/
  ...
www/         # 公开区域
  images/
  css/
  js/
  index.php
```

在这种情况下，项目根目录是symfony/目录。所以index.php前端控制器只要这样定义整个应用程序就可以工作了：

```
define('SF_ROOT_DIR', dirname(__FILE__).'/../symfony');
```

第19章将会告诉你更多如何修改symfony来实现特殊的目录结构的信息。

主应用程序配置

主应用程序配置存放在myproject/apps/myapp/config/目录下的文件里：

- app.yml：这个文件存放应用程序相关的配置信息，包括定义业务或者程序逻辑的全局变量，这些都不需要存放在数据库里。税率，运费，e-mail地址等经常存放在这个文件。这个文件默认是空的。
- config.php：这个文件引导应用程序，它会做所有的基础初始化来启动应用程序。这里你可以定制目录结构或者增加应用程序相关的常量(详见第19章)。他首先会包含项目的config.php文件。
- factories.yml：symfony在这里定义处理视图、请求、回应、会话(session)等的类。如果你想用你自己的类取代symfony的类，你可以在这里定义它们。详见第19章。
- filters.yml：过滤器是在每个请求都被执行的一小段代码。这个文件用来定义哪些过滤器需要被执行，每个模块都可以改写过滤器配置。详见第6章。
- logging.yml：这个文件定义哪些情况需要被记录到日志里，从而管理与调试应用程序。详见第16章。
- routing.yml：路由规则，能把难以理解的不好记忆的URL变成"漂亮"的直观形式。此配置文件用来存放这些信息。每个新应用程序都会有一些默认路由规则。详见第9章。

- settings.yml：这个文件存放 symfony 应用程序的主要配置信息。你的应用程序是否使用国际化功能，它的默认语言，请求 timeout 时间，是否开启缓存功能等都在这个文件里面定义。只要改变这个文件里的一行代码，你就可以关闭网站来执行维护升级。这些设置还有它们的用法详见第 19 章。
- view.yml：这个文件里定义默认视图的结构(布局的名称，标题，还有 meta tag；默认载入的样式表及 Javascript；默认的 content-type 等)。还有默认的 meta 与标题标签。详见第 7 章。这些配置信息可以在模块里改写。

国际化配置

国际化的应用程序可以显示多种语言。这需要特殊的配置。国际化配置信息存放在如下两个地方：

- 应用程序 config/目录里的 i18n.yml：这个文件定义一般的翻译设置，例如原文的语言、在数据库还是文件里面存放翻译信息还有翻译信息的格式等。
- 应用程序 i18n/目录里的翻译文件：这些文件基本上是字典，里面包含程序模板里面出现的所有文字的翻译，这样切换语言的地方就会显示对应的翻译。

注意开启 i18n(国际化)功能需要在 setting.yml 文件里面设置。详见第 13 章。

其它应用程序配置

还有一部分配置文件在 symfony 的安装目录里(在 $sf_symfony_data_dir/config/)，这些文件在所有的项目配置目录里都找不到。这是一些很少需要修改或者对于全部项目共用的配置信息。不过，如果你需要改动它们，只要在你的 myproject/apps/myapp/config/里建立一个相同名字的空文件，然后在里面改写你要修改的配置信息就可以了。应用程序里的配置信息总是优先于框架的配置信息。下面是 symfony 安装目录的 config/目录里的文件：

- autoload.yml；此文件包含了自动载入功能的配置信息。这个功能帮你从特定的目录自动载入你写的类。详见第 19 章。
- constants.php：此文件包含了默认的应用程序文件结构。请使用应用程序的 config.php 来覆盖这些配置信息。详见第 19 章。
- core_compile.yml 和 bootstrap_compile.yml：这两个文件记录启动应用程序(在 bootstrap_compile.yml 里)和处理请求(在 core_compile.yml 里)需要载入哪些类。这些类被压缩在一个没有注释的 PHP 文件里，这样可以最小化文件处理从而加快执行速度(每个请求只载入 1 个文件而不是 40 多个文件)。这在没有安装 PHP 加速器的时候特别有效。优化技术详见第 18 章。

- config_handlers.yml：在这个文件里你可以增加或者修改处理配置文件的工具。详见第 19 章。
- php.yml：这个文件用来检查 php.ini 里的配置信息是否满足程序需要，并且可以帮你覆盖 php.ini 的配置。详见第 19 章。

模块配置

默认情况下，模块没有特别的配置信息。不过，如果你需要，你可以为某个模块覆盖应用程序级的配置。例如，你需要修改一个模块里所有动作的 HTML 描述信息，或是载入一个特定的 Javascript 文件。你可以选择针对某个特定的模块增加新参数来实现保护性封装。

可能你已经猜到，模块配置文件必须放在 myproject/apps/myapp/modules/mymodule/config/目录里，模块配置信息有下面这几个文件：

- generator.yml：根据数据库表自动生成的模块(脚手架与管理后台)会用这个文件，它用来定义界面怎么显示行和列，用户可以执行哪些操作(过滤器，排序，按钮等)。详见第 14 章。
- module.yml：这个文件包含模块的特殊参数(相当于 app.yml，但这是模块级的),还有动作的配置信息。详见第 6 章。
- security.yml：这个文件用来给动作设置访问限制。你可以在这里设置哪个页面只能给注册用户看或是一部分有特殊权限的注册用户看。详见第 6 章。
- view.yml：这个文件包含模块的一个或者所有动作的视图配置信息。它会覆盖应用程序级的 view.yml，详见第 7 章。
- 数据验证文件：虽然这些用来验证表单输入数据的 YAML 数据验证文件存放在 validate/目录而不是 config/目录里，它们仍然属于模块配置文件。详见第 10 章。

大多数模块配置文件能让你为一个模块的所有视图或者动作定义参数，也可以只为模块里的一部份视图或者动作定义参数。

SIDEBAR 文件太多了？

可能应用程序里的配置文件太多了，使你受到了打击。不过请注意：

大多数时候你都不用修改配置，因为默认的配置就能够满足大部分的需求。每个配置文件与一个特定的功能相关，以后的章节会一个一个详细介绍它们的用法。当你关注某一个配置文件的时候，你就能清楚的了解它的用途还有它的组织形式。对于专业 web 开发，默认的配置不会经常被完全重写。配置文件使你不用修改一行代码轻易地改变 symfony 的功能。想想看要达到同样的目的需要多少 PHP 代码吧。如果所有的配置文件都存放在同一个文件里，那么这个文件不仅仅会变

得完全无法阅读，同样你也无法在不同的级别重新定义配置信息(请看本章后面的"配置层叠"小节)。 配置系统是 symfony 重大优点之一，它使 symfony 适合于几乎所有类型的 web 应用程序，不仅限于这个框架原本的设计目的。

环境

在开发应用程序的过程中，你可能同时需要好几套配置信息。例如，开发过程中用来测试的数据库配置信息，还有生产环境中正式的数据库配置信息。symfony 为满足同时使用不同配置信息的需求，提供了不同的环境。

什么是环境？

一个应用程序可以在不同的环境中运行。不同的环境共享相同的 PHP 代码(前端控制器除外)，但是配置信息可能完全不同。symfony 为每个应用程序提供三种默认的环境：生产(prod)、测试(test)和开发(dev)。只要你愿意你可以无限制的增加环境的数量。

所以基本上，环境与配置是同义词。例如，测试环境会记录警告与错误，生产(prod)环境只记录错误。缓存加速功能在开发(dev)环境下通常是关闭的，但是在测试(test)与生产(prod)环境下开启。开发与测试环境所需要的测试数据，存放在与生产环境不同的数据库里。所以两种环境的数据库配置会有所不同。所有的环境可以在一台机器上共存，不过通常一台正式的服务器只包含生产(prod)环境。

在开发环境下，日志与调试功能都是开启的，因为排除问题比性能更重要。相反，生产环境的配置默认是为性能优化的，所以生产环境会关闭一些功能。建议呆在开发环境直到你对开发的功能满意为止，然后切换到生产环境测试速度。

测试环境又与开发与生产环境有所不同。你只能通过命令行来访问这个环境，通常是做功能测试和执行批处理脚本。因此，测试环境与生产环境接近，但是不能通过浏览器访问。它能模拟 coookie 与其它 HTTP 相关的组件。

改变你正在访问的应用程序的环境，只要改变前端控制器就可以了。到目前为止，你只见过开发环境，因为例子里的 URL 都是开发环境前端控制器的：

http://localhost/myapp_dev.php/mymodule/index

不过，如果你想看看应用程序在生产环境中的样子，可以执行生产环境的前端控制器：

http://localhost/index.php/mymodule/index

如果你的 web 服务器支持 mod_rewrite，你甚至可以在 web/.htaccess 里自定义 symfony 重写规则。这些规则把生产环境的前端控制器作为默认的执行脚本能让 URL 看上去像这样：

http://localhost/mymodule/index

SIDEBAR 环境与服务器

不要把环境与服务器的概念混淆了。在 symfony 里，不同的环境是指不同的配置信息，对应不同的前端控制器(执行请求的脚本)。不同的服务器对应 URL 里的不同域名。

http://localhost/myapp_dev.php/mymodule/index

服务器　环境

通常，开发人员在一台开发服务器上工作，这台服务器不与互联网相连，所有的服务与 PHP 配置文件都可以自由修改。到了发布应用程序的时候，程序文件会传到生产服务器上然后最终用户才能访问得到。

这意味着同一台服务器上可以有很多个环境。例如，你甚至可以在你的开发服务器上运行生产环境。不过，大多数时候，生产服务器上只能访问生产环境，这样可以避免服务器的配置信息泄露以减少安全方面的风险。

定义一个新环境，不需要建立目录或者使用 symfony 命令行工具。只要建立一个新的前端控制器，修改这个前端控制器里面定义的环境名就可以了。这个环境会继承所有默认配置信息和所有环境的共同配置信息。下一章会有详细介绍。

配置层叠

同样的配置信息可以在不同的地方定义多次。例如，你想把你的程序的所有页面的 mime-type 定义成 text/html，只有一个 rss 模块例外，这个模块需要 text/xml 的 mime-type。symfony 可以让你在 myapp/config/view.yml 里面写第一个定义，然后在 myapp/modules/rss/config/view.yml 里面写第二个定义。配置系统了解模块级别的定义一定要覆盖应用程序级的定义。

实际上，symfony 具有如下的配置级别：

- 粒度级别：
    - 框架里的默认配置
    - 整个项目的全局配置 (在 myproject/config/ 里)

    - 项目中应用程序的配置（在 myproject/apps/myapp/config/ 里）
    - 模块的配置（在 myproject/apps/myapp/modules/mymodule/config/ 里）
- 环境级别：
    - 针对某一个环境
    - 所有环境

所有可以自定义的属性里，有一些是与环境有关的。因此，很多 YAML 配置文件是按照环境分成了好几段，最后一段针对所有环境。因此一个典型的 symfony 配置文件类似于例 5-12。

例 5-12 - symfony 配置文件的结构

```
# 生产环境设置
prod:
  ...

# 开发环境设置
dev:
  ...

# 测试环境设置
test:
  ...

# 自定义环境设置
myenv:
  ...

# 所有环境的设置
all:
  ...
```

另外，symfony 框架本身定义的默认值并不在项目的目录里，它们在你的 symfony 的$sf_syfmony_data_dir/config/目录里。默认的配置信息在例 5-13 的文件里设置。所有的应用程序都会继承到这些设置。

例 5-13 - 默认配置信息，在 $sf_symfony_data_dir/config/settings.yml 里

```
# Default settings:
default:
  default_module:         default
  default_action:         index
  ...
```

这些配置会在项目，应用程序，模块的配置信息里面以注释的形式反复出现，如例 5-14 所示，这样你就可以知道这些默认值并且可以修改它们。

例 5-14 - 默认配置信息， 在 myapp/config/settings.yml 中出现以供参考

```
#all:
 #  default_module:         default
 #  default_action:         index
 ...
```

这意味着一个属性可以多次定义，最后的取值取决于层叠的结构。任何一个特定环境里定义的参数优先于所有环境里定义的参数，所有环境里的参数优先于默认配置。模块配置里的参数优先于应用程序级里定义的同样参数，应用程序里的参数优先于项目级。这可以通过下面的优先级列表来表示：

1. 模块
2. 应用程序
3. 项目
4. 特定的环境
5. 所有环境
6. 默认

配置缓存

执行时解析 YAML 处理配置文件的层叠结构会增加每次请求的负担。symfony 内建了配置文件缓存机制来提高请求速度。

不管什么格式的配置文件都需要一些特别的类来处理，又叫处理器，这些配置文件被转换成快速执行的 PHP 代码。开发环境里，处理着每次请求都会去检查配置文件的变化，这样提高交互性。它们解析改变的配置文件使你能马上看到 YAML 改变的效果。但是在生产环境，这样的处理只在第一次请求时进行，处理得到的 PHP 代码被保存下来给后面的请求使用。这样能提高性能，因为生产环境里的每次请求只需要执行一些优化过的 PHP 代码。

例如，如果 app.yml 文件内容如下：

```
all:                    # 所有环境的设置
  mail:
```

```
    webmaster:          webmaster@example.com
```

那么 cache/目录下的 config_app.yml.php 文件,将包括下面的内容：

```
<?php

sfConfig::add(array(
  'app_mail_webmaster' => 'webmaster@example.com',
));
```

这样，大多数时候，symfony 不需要去解析 YAML 文件，只需要执行 cache 里的配置信息就可以了。不过在开发环境,symfony 会自动比较 YAML 文件还有配置缓存的修改时间，只重新处理上次请求后修改过的配置文件。

这是 symfony 与其他很多 PHP 框架相比的一个主要优势，这些 PHP 框架里的每次请求都会去处理配置文件，即使是生产环境。与 Java 不同，PHP 不会在请求之间共享执行状态。 其他依赖 XML 配置文件的框架在每次请求时处理 XML 性能损失很大。symfony 不存在这个问题，配置文件带来的速度影响很小。

这样的机制带来一个重要的问题，如果你改变了生产环境的配置信息，你需要强制重新解析所有你修改过的配置文件，这样改变才能生效。你只需要清除缓存就可以了，可以直接清空 cache/目录的内容，或是执行 clear-cache 这个 symfony 任务：

```
> symfony clear-cache
```

从代码里访问配置信息

所有的配置文件最终都被转换成 PHP 代码，框架会自动使用很多设置。不过，有时你需要在你的代码中(动作，模板，自定义类等)访问配置文件里定义的设置。settings.yml、apps.yml、module.yml、logging.yml 还有 i18n.yml 里的配置信息可以通过一个特殊的 sfConfig 类来访问。

sfConfig 类

你可以在程序代码里通过 sfConfig 类访问配置信息。它是一个配置信息的登记处，它有一些简单的存取方法，这些存取方法可以在程序的任何地方使用。

```
// 取得一个设置
parameter = sfConfig::get('param_name', $default_value);
```

注意你也可以在 PHP 代码里定义或者覆盖一个设置：

```
// 定义一个设置
sfConfig::set('param_name', $value);
```

参数的名字由几部分组成，中间用下划线分割，顺序如下：

- 与配置文件名有关的前缀 (sf_ 代表 settings.yml, app_ 代表 app.yml, mod_ 代表 module.yml, sf_i18n_ 代表 i18n.yml, sf_logging_ 代表 logging.yml)
- 父键名 (如果有), 小写形式
- 键名, 小写形式

参数名字不包括环境名称，因为PHP代码只能访问到执行时所在的环境里定义的参数。

例如，如果你需要访问app.yml里定义的值，见例5-15，你需要例5-16中的代码。

例 5-15 - app.yml 配置文件样本

```
all:
  version:        1.5
  .general:
    tax:          19.6
  default_user:
    name:         John Doe
  mail:
    webmaster:    webmaster@example.com
    contact:      contact@example.com
dev:
  mail:
    webmaster:    dummy@example.com
    contact:      dummy@example.com
```

例 5-16 - 在dev环境从PHP代码里访问配置信息

```
echo sfConfig::get('app_version');
 => '1.5'
echo sfConfig::get('app_tax');   // 请注意分类的头会被忽略掉
 => '19.6'
echo sfConfig::get('app_default_user_name);
 => 'John Doe'
echo sfConfig::get('app_mail_webmaster');
 => 'dummy@example.com'
echo sfConfig::get('app_mail_contact');
```

```
 => 'dummy@example.com'
```

所以 symfony 的配置信息有所有 PHP 常量的优点，但是没有 PHP 常量的缺点，因为 symfony 配置的值可以改变。

所以, 用来给应用程序设置框架设置的 settings.yml 文件,相当于一系列的 sfConfig::set()调用。例 5-17 会被解释为 例 5-18。

例 5-17 - 不完整的 settings.yml

```
all:
  .settings:
    available:              on
    path_info_array:        SERVER
    path_info_key:          PATH_INFO
    url_format:             PATH
```

例 5-18 - symfony 处理 settings.yml 文件的结果

```
sfConfig::add(array(
  'sf_available' => true,
  'sf_path_info_array' => 'SERVER',
  'sf_path_info_key' => 'PATH_INFO',
  'sf_url_format' => 'PATH',
));
```

settings.yml 文件里面设置的含义请参考第 19 章。

自定义应用程序配置与 app.yml

大部分与程序功能有关的设置存放在 app.yml 文件里，这个文件在 myproject/apps/myapp/config/目录。app.yml 与环境有关，默认是空的。把所有你需要很容易修改的设置放在这个文件里，在代码里用 sfConfig 类访问它们。如例 5-19。

例 5-19 - 这个 app.yml 给指定的网站定义接受的信用卡类型

```
all:
  creditcards:
    fake:             off
    visa:             on
    americanexpress:  on

dev:
```

```
  creditcards:
    fake:             on
```

想要知道当前环境是否接受 fake 信用卡，需要这么写代码：

```
sfConfig::get('app_creditcards_fake');
```

TIP 当你要定义一个常量或者一个设置的时候，考虑一下把它放在 app.yml 里面会不会更好。在这里存放应用程序配置很方便。

如果你的自定义参数用 app.yml 的语法难以处理，你可以考虑自己定义一套语法。这样你可以把配置信息存在一个新的文件里，用新的处理器解析配置文件。配置文件处理器的资料详见第 19 章。

更多使用配置文件的技巧

在开始写你自己的 YAML 文件之前，有一些最后的技巧需要掌握。这些技巧可以避免配置信息的重复及其如何处理你自己的 YAML 格式。

在 YAML 文件里使用常量

一些配置设置的值取决于其他的设置。为了避免重复设置同样的值，symfony 支持在 YAML 文件里使用常量。如果遇到%包起来的大写形式的设置名(可以通过 sfConfig::get()取得值)，配置文件处理器会用这个设置的当前值来替换这个常量。见例 5-20。

例 5-20 - 在 YAML 文件里使用常量，以 autoload.yml 为例

```
autoload:
  symfony:
    name:           symfony
    path:           %SF_SYMFONY_LIB_DIR%
    recursive:      on
    exclude:        [vendor]
```

path 参数的值会是执行 sfConfig::get('sf_symfony_lib_dir')的结果。如果一个配置文件依赖于另一个配置文件，被依赖的配置文件必须先被解析(请查看 symfony 的源代码来了解配置文件载入的顺序)。app.yml 是最后被解析的文件之一，所以你可以在这个文件里使用其它文件里定义的设置。

在配置文件里使用脚本

有可能你的配置信息与外部参数有关(例如数据库或者其他配置文件)。为了解决这种问题，symfony 在把配置文件传给 YAML 处理器之前先用 PHP 来解析配置文件。这意味着你可以在 YAML 文件里使用 PHP 代码，如例 5-21。

例 5-21 - YAML 文件可以包含 PHP

```
all:
  translation:
    format:  <?php echo sfConfig::get('sf_i18n') == true ? 'xliff' :
'none' ?>
```

但是请注意配置文件在请求周期早期就被处理了，所以你不能使用 symfony 内建的方法或者函数。

CAUTION 在生产环境中，配置文件会被缓存，所以配置文件只会在清除缓存后被处理(并执行)一次。

浏览你的 YAML 文件

如果你想直接读取 YAML 文件，你可以使用 sfYaml 类。它是一个可以把 YAML 文件转化成 PHP 数组的 YAML 解析器。例 5-22 是一个 YAML 文件的例子，例 5-23 是前面 YAML 文件的解析结果。

例 5-22 - test.yml

```
house:
  family:
    name:     Doe
    parents:  [John, Jane]
    children: [Paul, Mark, Simone]
  address:
    number:   34
    street:   Main Street
    city:     Nowheretown
    zipcode:  12345
```

例 5-23 - 使用 sfYaml 类把 YAML 转换成一个数组

```
$test = sfYaml::load('/path/to/test.yml');
print_r($test);

Array(
  [house] => Array(
    [family] => Array(
```

```
      [name] => Doe
      [parents] => Array(
        [0] => John
        [1] => Jane
      )
      [children] => Array(
        [0] => Paul
        [1] => Mark
        [2] => Simone
      )
    )
    [address] => Array(
      [number] => 34
      [street] => Main Street
      [city] => Nowheretown
      [zipcode] => 12345
    )
  )
)
```

总结

symfony 配置系统使用的 YAML 语言简单并且可读性强。多种环境与层叠结构的参数定义为开发者提供了多种选择。一些配置文件可以从代码里通过 sfConfig 类访问，特别是 app.yml 里的应用程序配置。

的确,symfony 有很多配置文件，不过这使 symfony 适用性更强。注意除非你的应用程序需要高级别的定制，否则你根本不需要去修改它们。