ORQUESTRANDO APLICAÇÕES PHP COM SYMFONY

Flávio Gomes da Silva Lisboa

Novatec

Editor: Rubens Prates
Assistente editorial: Priscila A. Yoshimatsu
Revisão gramatical: Smirna Cavalheiro
Capa: Carolina Kuwabata
Editoração eletrônica: Carolina Kuwabata

ISBN: 978-85-7522-464-9

Histórico de impressões:

Novembro/2015 Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Email: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec
LinkedIn: linkedin.com/in/novatec

CAPÍTULO 1

Introdução

"Uma sinfonia deve ser como o mundo. Precisa conter tudo."

– Gustav Mahler

Um maestro não precisa saber tocar todos os instrumentos para reger uma orquestra. Ele precisa apenas conhecer o potencial de cada instrumento. Conhecer o potencial significa saber o que ele é capaz de fazer. Entretanto, existe algo que o maestro conhece em profundidade: os fundamentos da teoria musical.

Da mesma forma, não é possível falar sobre um framework que trabalha com programação avançada em PHP usando orientação a objetos sem saber como programar como a implementação de orientação a objetos em PHP. Sendo mais objetivo, o aproveitamento da leitura deste livro depende do domínio de alguns fundamentos: HTML, linguagem de programação PHP e orientação a objetos (conceitos de classe, herança, encapsulamento, polimorfismo). Consideramos implícita a necessidade de conhecer o sistema operacional do computador que você utilizará para implementar os exemplos deste livro.

Criar aplicações sem utilizar bancos de dados não faz parte de nossos objetivos. Por isso você precisa ter conhecimentos básicos sobre bancos de dados relacionais e linguagem SQL. Embora nosso objetivo seja abstrair a comunicação com o banco de dados, você deverá compreender o que está ocorrendo atrás das cortinas.

Conforme afirmamos no princípio, o maestro precisa conhecer o potencial de cada instrumento. Ninguém sai regendo uma orquestra sinfônica da primeira vez. É preciso saber como cada instrumento se sai como solista, depois conhecer as combinações possíveis entre os instrumentos de cada

classe (cordas, madeiras, metais, percussão e teclas) até se chegar a uma combinação de todos os instrumentos.

Asim, propomos que por meio deste livro você adquira gradualmente o conhecimento sobre a construção de uma aplicação PHP, cuja estrutura será provida pelos componentes do framework Symfony. Começaremos conhecendo o solo de cada componente, ou seja, o uso de um componente de forma desacoplada do resto da biblioteca, e adicionaremos cada vez mais componentes até que utilizemos o Symfony inclusive para implementar as três camadas do padrão de projeto MVC.

1.1 Programação orientada a objetos em PHP

Symfony é implementado com programação orientada a objetos. Por isso precisamos compreender como funciona a implementação de orientação a objetos do PHP antes de falar sobre o framework. Veremos aqui o estritamente necessário para não ficarmos perdidos quando começarmos a utilizar os componentes do Symfony. Você pode fazer uma leitura mais profunda sobre programação orientada a objetos em PHP consultando o livro *PHP – programando com orientação a objetos,* de Pablo Dall'Oglio.

Segundo Sebesta (2000, p. 418), "uma linguagem com esta característica" (orientação a objetos) "deve oferecer três recursos-chave: tipos de dados abstratos, herança e um tipo particular de vinculação dinâmica".

"Um tipo de dado abstrato", de acordo com Sebesta (2000, p. 398), "é um encapsulamento que inclui somente a representação de dados de um tipo específico de dado e os subprogramas que fornecem as operações para esse tipo".

"A herança", segundo Sebesta (2000, p. 419), "oferece uma solução tanto para o problema da modificação, apresentado pela reutilização de tipos de dados abstratos, como pelo problema de organização do programa". Sebesta (2000, p. 418) afirma que a herança "é o centro da programação orientada a objeto" e que ela consiste no seguinte: "os atributos comuns de uma coleção de tipos de dados abstratos similares são fatorados e colocados em um novo tipo. Os membros da coleção herdam as partes comuns deste."

Conforme Sebesta (2000, p. 421), a vinculação dinâmica na programação orientada a objetos permite que "sistemas sejam mais facilmente estendidos tanto durante o desenvolvimento como durante a manutenção".

PHP trabalha com vinculação dinâmica de tipos. Sebesta (2000, p. 166) esclarece o que vem a ser isso, dizendo o seguinte:

> Com a vinculação dinâmica de tipos, o tipo não é especificado por uma instrução de declaração. Em vez disso, a variável é vinculada a ele quando lhe é atribuído um valor em uma instrução de atribuição. Quando a instrução de atribuição é executada, a variável que está sendo atribuída é vinculada ao tipo do valor, da variável ou da expressão no lado direito da atribuição.

Isso significa que em PHP as variáveis têm o tipo definido no momento da atribuição do valor – e não há declaração de variáveis. Os tipos de variável do PHP são boolean, integer, floating point, string, array, resource e object. O tipo object é o único que não mantém apenas um valor escalar. Ele permite que o programador agrupe muitos tipos básicos do PHP com o nome que quiser.

Segundo Shafik e Ramsey (2006, p. 132), "a programação orientada a objetos gira ao redor do conceito de agrupamento de código e dados juntos em unidades lógicas chamadas classes". Uma classe em PHP é um bloco de código nomeado pela palavra-chave `class`. Segundo a convenção da PSR-1, nomes de classes usam letras maiúsculas para as iniciais de cada palavra que compõe o nome, mantendo as demais letras como minúsculas.

> **NOTA:** PSR-1 é um padrão de codificação básica em PHP criado pelo Framework Interoperability Group (PHP-FIG). Os padrões do PHP-FIG estão publicados em *www.php-fig.org*.

A seguir a estrutura básica de uma classe:

```
class ClassName
{
}
```

Objetos são criados com o operador `new`. Esse operador retorna uma referência ao objeto criado, que pode ser armazenada em uma variável. Como o acesso a objetos é feito por referência, para copiar um objeto é preciso usar o operador clone.

Por exemplo, no código a seguir:

```
$someObject = new ClassName();
$sameObject = $someObject;
```

As variáveis `$someObject` e `$sameObject` apontam para o mesmo objeto. Para criar uma cópia do objeto referenciado por `$a`, temos de fazer de acordo com o seguinte código:

```
$someObject = new ClassName();
$otherObject = clone $someObject;
```

A comparação entre objetos pode ser feita de duas formas: por igualdade e por identidade. Quando se usa o operador `==` para comparar dois objetos, o resultado é baseado na igualdade de atributos e valores e na classe a que pertencem. Deste modo, dois objetos são iguais se são instâncias da mesma classe e seus atributos têm os mesmos valores.

Quando se usa o operador `===`, por outro lado, o resultado é restrito a referências à mesma instância. Isso significa que `$o1 === $o2` retornará `TRUE` se, e somente se, as variáveis `$o1` e `$o2` tiverem referências para o mesmo objeto.

Uma classe tem constantes atributos e métodos. No PHP, os atributos são variáveis e os métodos são funções. O que os diferencia de suas contrapartes da programação estruturada é a inclusão da visibilidade e do contexto estático de execução.

Existem três modificadores de visibilidade no PHP: private, protected e public. Atributos e métodos identificados com o primeiro são visíveis somente pela classe que os declarou. Os que são identificados com protected são visíveis pela classe e suas herdeiras. O modificador public permite a visibilidade para qualquer classe.

```
class ClassName {
   private $firstAttribute;
   protected $secondAttribute;
   public $thirdAttribute;
   public function getFirstAttribute()
   {
   }
}
```

A princípio, os atributos e métodos pertencem aos objetos. Para armazenar um dado em um atributo, é preciso haver um objeto em memória. Para executar um método, é preciso haver um objeto por meio do qual o invoquemos. No contexto estático, os atributos e métodos são da classe e não do objeto. Isso permite armazenar dados que podem ser compartilhados entre vários objetos da mesma classe e executar métodos sem a necessidade de instanciar uma classe. O contexto estático é definido pela palavra-chave `static`.

```
class NomeDaClasse {
    private static $atributo1;
    public static function getAtributo1()
    {
    }
}
```

A ferramenta de reuso da orientação a objetos é a herança. Lembraremos disso mais adiante quando herdarmos de várias classes do Symfony. A herança consiste em identificar código-fonte genérico e deslocá-lo para uma classe abstrata. A classe abstrata não deve ser instanciada, mas estendida por outra, que implementará os métodos abstratos e adicionar outros métodos específicos.

```
abstract class ClasseGenerica {
    protected function metodoGenerico() {
    }
    abstract public function metodoEspecifico();
}

class ClasseEspecifica extends ClasseGenerica {
    public function metodoEspecifico() {
    }
}
```

O oposto da classe abstrata é a classe final. Ao colocar a palavra "final" antes do nome de uma classe, você a esteriliza: ela não pode ter herdeiras.

Métodos abstratos podem ser concentrados em interfaces, que são construções que contêm somente assinaturas de métodos. Uma classe pode herdar somente uma classe, mas pode implementar várias interfaces. Isso permite definir comportamento padronizado para várias classes.

```
interface InterfaceGenerica {
   public function metodoPadrao();
}
class ClasseEspecifica implements InterfaceGenerica {
   public function metodoPadrao() {
   }
}
```

Interfaces definem um padrão de assinatura para métodos, mas não os implementam. A partir da versão 5.4, PHP disponibilizou os traits, que são blocos de código reutilizáveis. Traits implementam atributos e métodos e são utilizados para compartilhar código que não pode ser centralizado em classes abstratas.

```
trait TraitEspecifico {
   public function metodoEspecifico() {
   // implementação
   }
}

class ClasseEspecifica extends ClasseGenerica implements InterfaceGenerica
{
   use TraitEspecifico;
}
```

Para evitar a colisão de nomes, que é a impossibilidade de convivência em uma aplicação de duas classes com o mesmo nome, a partir da versão 5.3 o PHP implementou namespaces. Um mesmo nome de classe pode existir ao mesmo tempo, desde que ligado a namespaces diferentes. Para definir um namespace, colocamos a seguinte instrução na primeira linha de um arquivo que contém uma classe:

```
namespace Pacote1\Pacote2\Pacote3;
```

Se o nome da classe for `Classe1`, o nome completamente qualificado da classe será `Pacote1\Pacote2\Pacote3\Classe1`. Para que não seja necessário utilizar o nome completo, que é muito extenso, usamos o operador `use` antes da declaração da classe, para informar que estamos utilizando as classes de um determinado namespace:

```
use  Pacote1\Pacote2\Pacote3;
```

Não é necessário usar este comando se este já for o namespace da classe. O objetivo é acessar outros namespaces. Se uma classe precisa utilizar duas classes homônimas, ela tem duas opções. A primeira é referenciar o nome completamente qualificado assim:

```
$objeto1 = new Pacote1\Pacote2\Pacote3\Classe1();
$objeto2 = new PacoteA\PacoteB\PacoteC\Classe1();
```

A segunda é criar um apelido para uma das classes antes da declaração da classe que as referencia:

```
use  PacoteA\PacoteB\PacoteC\Classe1 as Classe1deB;
```

De modo que a instanciação seja feita pelo apelido:

```
$objeto2 = new Classe1deB();
```

Assim, não há colisão de nomes. É possível ter inúmeras classes com o mesmo nome, mas cada uma em um namespace diferente.

O PHP contém métodos mágicos, que não são invocados explicitamente, mas disparados por determinados eventos relacionados ao objeto em questão. A partir da versão 5.3.0, são eles:

- `__construct()` – Este método é chamado quando o objeto é criado. Usamos este método para configurar valores iniciais do objeto, inclusive por meio de argumentos passados.
- `__destruct()` – Este método é chamado quando o objeto é destruído.
- `__call()` – Este método é chamado quando um método inexistente é invocado. Isso permite criar métodos virtuais.
- `__callStatic()` – Similar a `__call()`, é chamado quando um método estático inexistente é invocado.
- `__get()` – Este método é chamado quando se tenta ler um atributo inexistente.
- `__set()` – Este método é chamado quando se tenta gravar em um atributo inexistente.
- `__isset()` – Este método é chamado quando a função `isset()` ou `empty()` é usada para atributos inexistentes.

- `__unset()` – Este método é chamado quando o método `unset()` é usado para um atributo inexistente.
- `__sleep()` – Este método é chamado quando um objeto é serializado.
- `__wakeUp()` – Este método é chamado quando uma serialização é revertida.
- `__toString()` – Este método é chamado quando se tenta imprimir um objeto (enviá-lo para a saída).
- `__invoke()` – Este método é chamado quando se tenta chamar um objeto como uma função.
- `__set_state()` – Este método é chamado quando a função `var_export()` é chamada ao receber um objeto como argumento.
- `__clone()` – É chamado quando o operador `clone` é usado sobre o objeto. Isso permite alterar o clone.

Ao desenvolver com Symfony lidaremos com a maioria dessas construções. Na verdade, Symfony não somente usa a orientação objetos para implementar o seu negócio, como parte do seu negócio é orientação a objetos.

1.2 Requisitos

A versão 2.7.4 do Symfony exige no mínimo PHP 5.4. É necessário dominar o sistema operacional do computador utilizado no livro para instalar os programas necessários à execução do Symfony. Isso significa que este é um livro sobre Symfony e não sobre seu sistema operacional. Não é possível fazer um curso de equitação sem cavalo.

1.3 Instalação do ambiente de desenvolvimento

Precisamos definir um ambiente de desenvolvimento com um conjunto mínimo de ferramentas para uma boa execução de nossas atividades. Orientamos a seguir sobre os procedimentos para instalar os softwares usados para criar os exemplos descritos no livro.

1.3.1 Apache, PHP e MySQL

Apache Friends é uma organização sem fins lucrativos, criada para promover o uso do servidor web Apache, por meio de atividades que tornam mais amigáveis a instalação de software e a leitura de documentação, além da criação de uma comunidade online para ajudar os usuários de Apache. O projeto foi criado em 2002 por Kai 'Oswald' Seidler e Kay Vogelgesang.

O principal produto gerado por essa organização é o XAMPP, que oferece distribuições para GNU/Linux (SuSE, RedHat, Mandriva e Debian), Windows (2000, XP, Vista e 7), Mac OS X e Solaris.

O XAMPP é um pacote de softwares que inclui principalmente Apache, MySQL, PHP e PERL (o X refere-se ao sistema operacional e as demais letras são iniciais desses softwares). O URL do projeto é: *http://www.apachefriends.org/pt_br/xampp.html* (Figura 1.1).

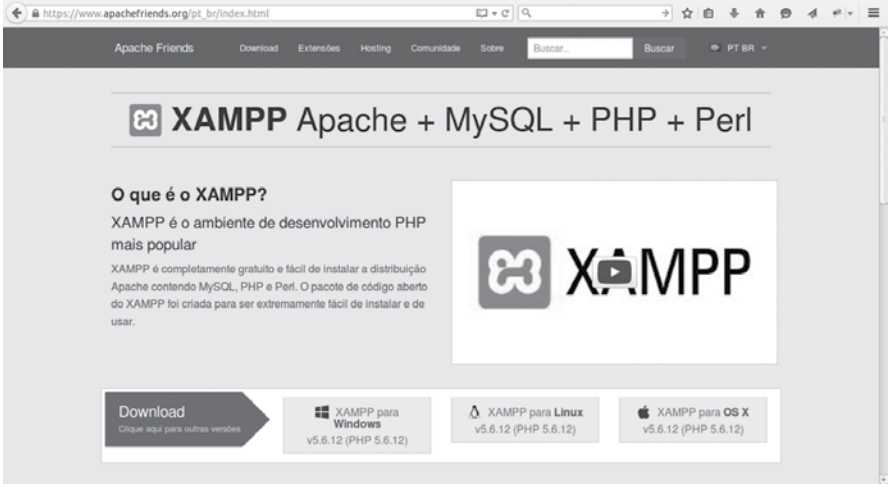

Figura 1.1 – Página do projeto Apachefriends, que mantém o XAMPP.

Tudo o que você precisa fazer é baixar o programa instalador relativo ao seu sistema operacional e executá-lo. Para facilitar a operação do XAMPP, o instalador inclui um painel de controle (Figura 1.2).

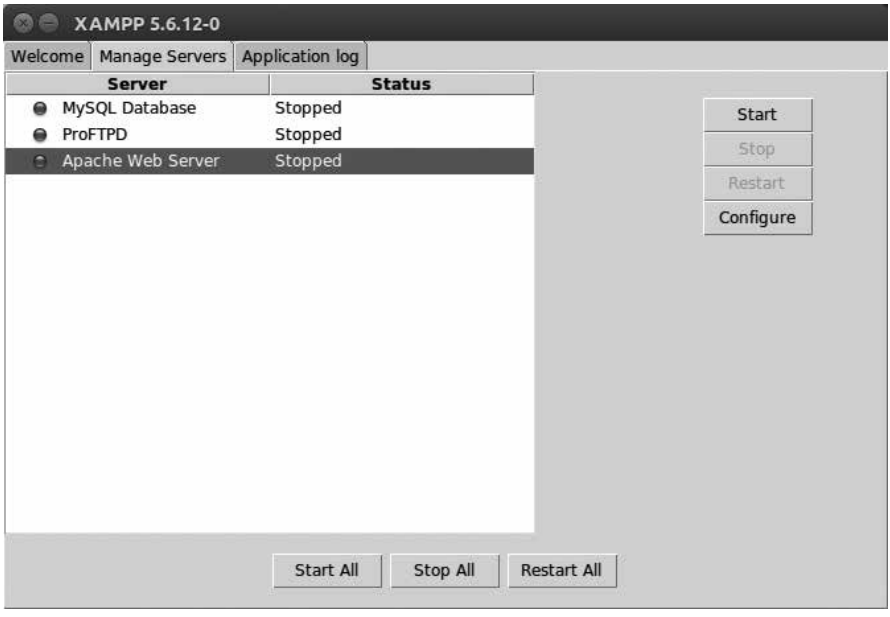

Figura 1.2 – Tela principal do painel de controle do XAMPP para GNU/Linux.

Se você já tiver um servidor web instalado, que tenha suporte à reescrita de URL, assim como um interpretador PHP com versão igual ou superior a 5.3.3 e um módulo que permita ao servidor usar o PHP, além de um banco de dados, não precisa necessariamente utilizar o XAMPP. A questão é que os exemplos deste livro foram construídos com esse pacote de softwares e só podemos garantir que funcionarão nesse ambiente, pois seria inviável tratar da miríade de possibilidades de instalação do quarteto de softwares que constitui o cerne do XAMPP.

A versão de XAMPP utilizada para criar e/ou executar os exemplos deste livro é a 5.6.12. No apêndice A você encontra mais informações sobre comandos do XAMPP.

1.3.2 NetBeans

NetBeans é um projeto de software livre que provê um ambiente integrado de desenvolvimento para Java, HTML 5, PHP, Groovy, C e C++. O editor de código-fonte do NetBeans é extremamente poderoso e oferece algumas facilidades para trabalhar com Symfony. Vamos baixar

a versão 8.0.2 da distribuição NetBeans para PHP a partir da página *https://netbeans.org/downloads/index.html*. Clique sobre o botão **Download** da coluna PHP (Figura 1.3).

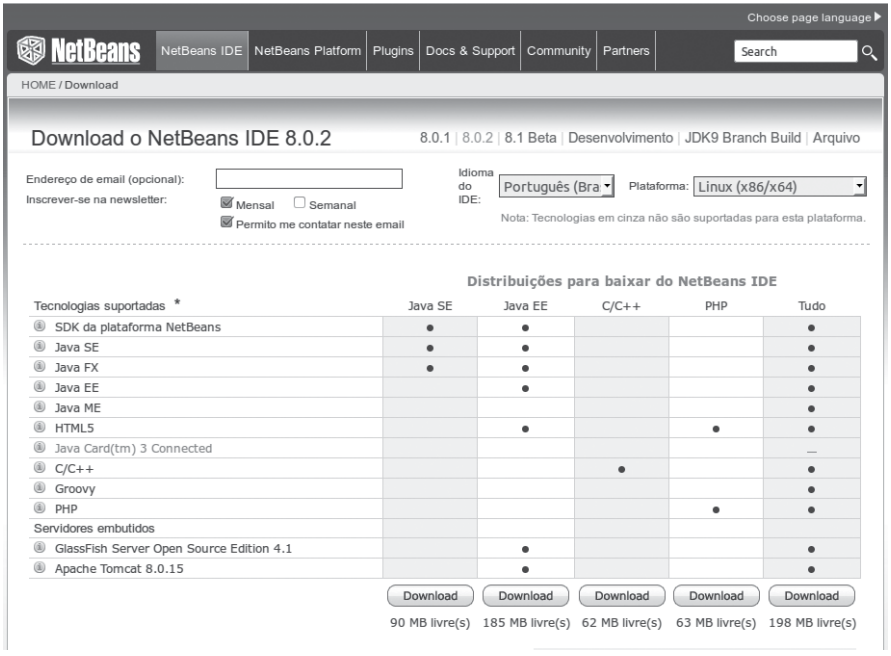

Figura 1.3 – Página de download das distribuições do NetBeans.

Será baixado um programa instalador, no formato executável do seu sistema operacional. Execute o programa. É necessária a máquina virtual Java (JVM) para instalar e executar o NetBeans. Java pode ser obtida em *https://java.com/en/download*.

O Symfony pode executar arquivos e projetos PHP, desde que seja associado a um executável do PHP. Isso é feito no menu **Ferramentas**, item **Opções**. Selecione o item **PHP** e a aba **Geral**, conforme mostra a figura 1.4. No campo **Interpretador do PHP 5** informe o caminho completo para o interpretador PHP.

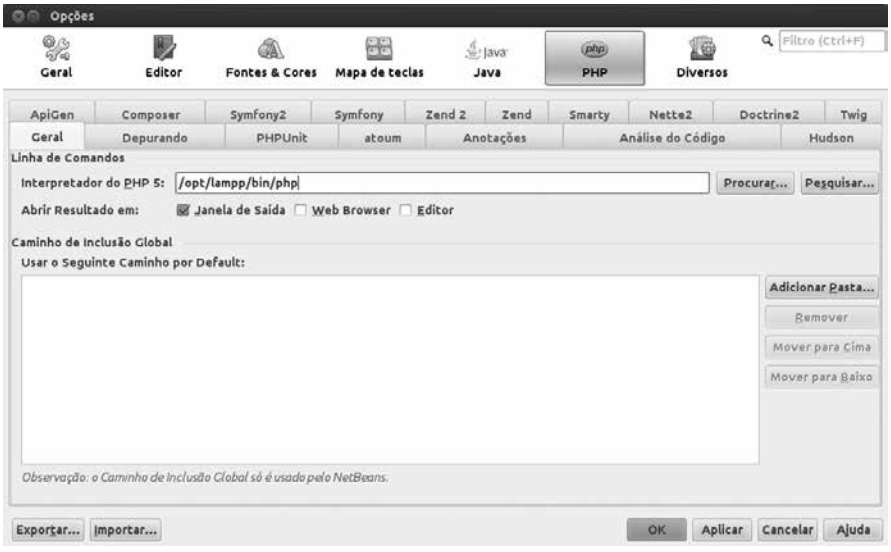

Figura 1.4 – Configuração do executável do PHP no NetBeans.

1.3.3 Symfony

Utilizaremos o release 2.7 do Symfony. Ele pode ser obtido no URL *http://symfony.com/download*.

O primeiro passo é baixar o instalador do Symfony. Você pode fazer isso com o seguinte procedimento:

- **GNU/Linux**

```
$ sudo curl -LsS http://symfony.com/installer -o /usr/local/bin/symfony
$ sudo chmod a+x /usr/local/bin/symfony
```

- **Windows**

```
c:\> php -r "readfile('http://symfony.com/installer');" > symfony
```

Ao final do procedimento, você deverá ter uma aplicação PHP chamada *symfony* disponível no caminho de busca do seu sistema operacional. No Windows é necessário executar o arquivo *symfony* com o interpretador `php` (assim: `php symfony`).

O instalador do Symfony é tendencioso. Ele foi feito para criar projetos que utilizarão a pilha completa de componentes do framework (ou ao menos os mais fortemente acoplados que ele tem). Por isso nós adiaremos o uso do instalador para o penúltimo capítulo, quando criaremos um projeto que utilizará o MVC o Symfony.

Não é nossa intenção convencê-lo a usar todos os componentes do Symfony, ou sua implementação de MVC (que acopla muitos deles). Você verá neste livro que pode usar simplesmente o que quiser, guardadas as devidas restrições, pois os componentes de mais alto nível, em sua maioria, são desacoplados.

Isso permitirá que você reflita sobre um modo mais simples de resolver um problema e um modo mais complexo, oculto sob abstrações. Se o modo complexo, encapsulado pelo Symfony, for satisfatório para você, ótimo. Senão, você saberá que tem opções, que pode realizar o mesmo processo sem o Symfony e até substituir um componente do Symfony por um componente similar de outro framework.

É muito importante alcançar o final deste livro consciente de que você tem opções, que tem alternativas. Por isso usaremos o Composer para baixar apenas os componentes Symfony que forem necessários em cada capítulo.

1.3.4 Composer

Composer é um gerenciador de dependências para projetos PHP. Você pode instalar a versão mais recente do aplicativo Composer de várias maneiras:

- **Usando o curl**:

```
curl -sS https://getcomposer.org/installer | php
```

- **Usando o PHP**:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

- **Baixando o arquivo diretamente**:

```
php -r "readfile('https://getcomposer.org/installer');" | php
```

O que importa é que você consiga chamar o comando `composer` ao final do procedimento, ou que consiga executar o comando `php composer.phar`.