Php 框架调研报告

目录

1.Phalcon

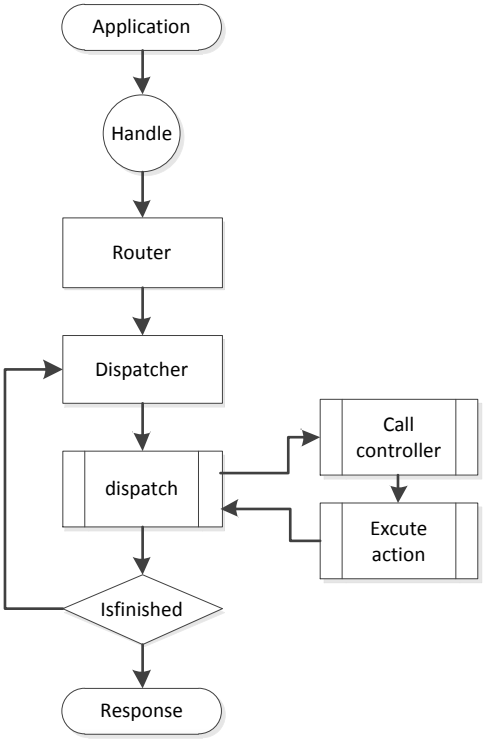
1.1 概述

Phalcon 是一个开源的，全堆栈的，用 C 语言写成的 php5 框架，专为高性能设计。你不需要学习和使用 C 语言的功能，因为这是一个 PHP 框架，只不过用 C 写成而已。同时 Phalcon 是松耦合的，您可以根据需要使用其他组件。

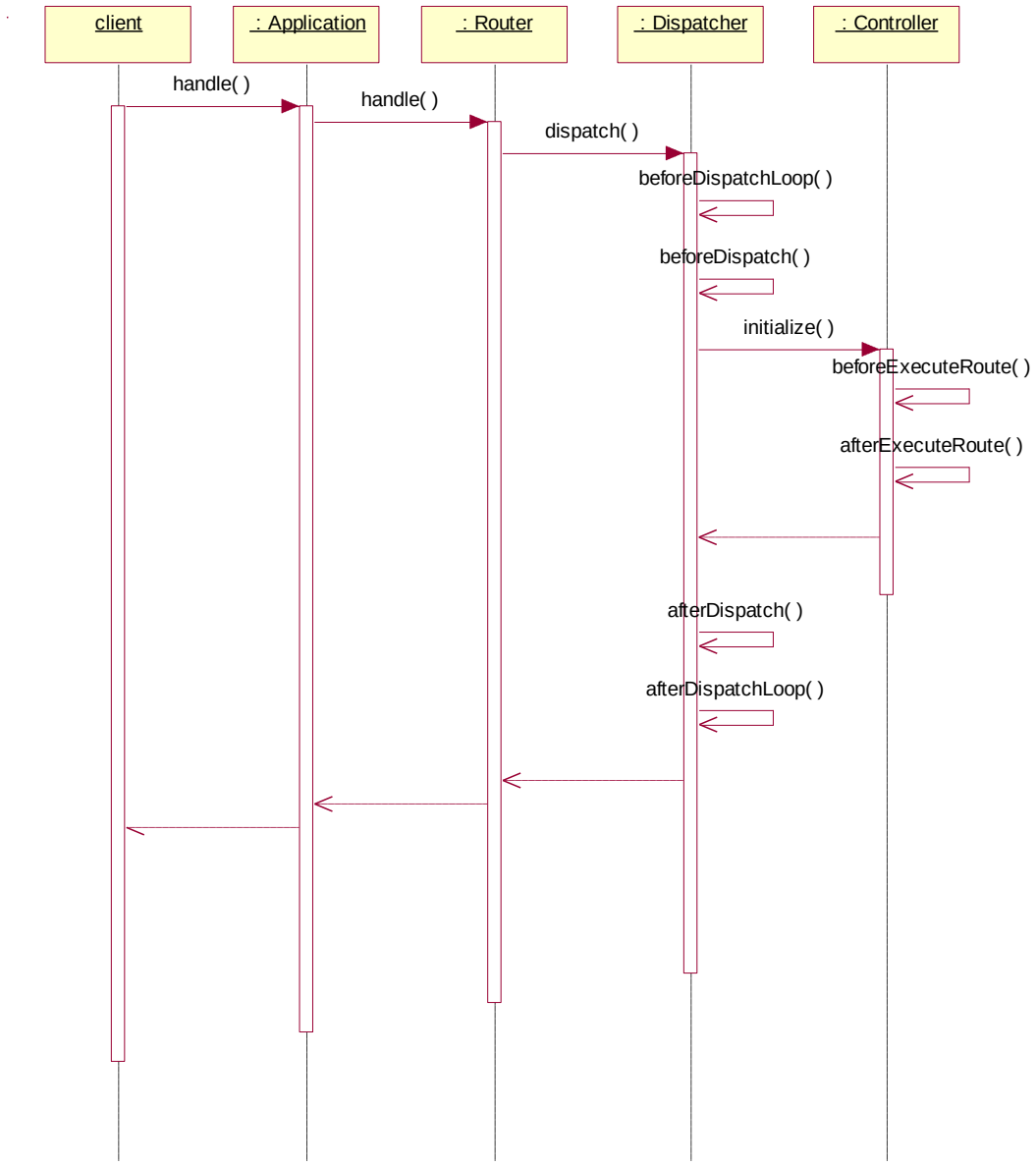
1.2 路由机制

Phalcon 的路由机制 router 定义用户请求对应到哪个控制器或 Action。router 解析 URI 以确定这些信息。路由器有两种模式：MVC 模式和匹配模式(match-only)，MVC 模式即用户明确指定什么样的 URI 对应什么样的 controller 及 action，如/users/add 则对应到 UsersController@addAction 等，这样需要用户为每一个 URI 都进行到 controller 及 action 的映射配置；而匹配模式是根据 URI 正则匹配规则，统一映射到相应的 controller 及 action，如/admin/:xxxxx/:yyyyy，这样的 URI 对对应到 admin model 下的 xxxxxController@yyyyyAction 那里。

1.2.1Phalcon 框架运行流程图

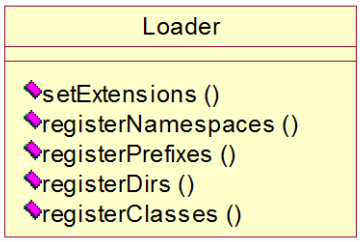


1.2.2 路由基本流程图



1.3 自动加载机制

Phalcon 的自动加载机制可以实现类的自动查找加载等功能。主要是由 Loader 类来实现，其中也是由 php 自带的函数 spl_autoload_register()来具体实现，其类图如下所示：



如图所示有几种方法

1、 setExtensions ()，该方法可以设置几种文件扩展名，Loader 依次根据这些扩展名进行查找并 include 类文件。

2、 registerNamespaces ()，注册命名空间及相对应的目录，如：

```
$loader->registerNamespaces(
    array(
        "Example\Base"      => "vendor/example/base/",
        "Example\Adapter" => "vendor/example/adapter/",
        "Example"             => "vendor/example/",
    )
);
```

3、 registerPrefixes()，注册一些类名的前缀与其对应的目录，即这些前缀的类应该在这些目录里查找，如：

```
$loader->registerPrefixes(
    array(
        "Example_Base"       => "vendor/example/base/",
        "Example_Adapter"  => "vendor/example/adapter/",
        "Example_"             => "vendor/example/",
    )
);
```

4、 registerDirs()，注册一些可以在这些目录里找类的目录，如：

```
$loader->registerDirs(
    array(
        "library/MyComponent/",
        "library/OtherComponent/Other/",
        "vendor/example/adapters/",
        "vendor/example/"
    )
);
```

5、 registerClasses()，该方法注册一些类及其所在的目录，如：

```
$loader->registerClasses(
    array(
        "Some"              => "library/OtherComponent/Other/Some.php",
        "Example\Base" => "vendor/example/adapters/Example/BaseClass.php",
    )
```

);

1.4controller

这里的控制器提供了一些方法，我们叫做 action，通过这些 action 方法来处理请求，并返回处理结果响应给用户。
所有的控制器都继承 Controller 这个基类，且必须以 Controller 做为后缀，方法以 Action 做为后缀，如：

```
class UserController extends \Phalcon\Mvc\Controller
{

    public function    addAction()
    {

    }
}
```

所有的控制器都会自动加载，根据注入的 controller 存入目录进行查找并 include，且继承 Controller 的控制器都可以直接调用获取一些服务进行应用，同时每个控制器里都会有 request 和 response 等请求响应对象供其调用。控制器也提供了 initialize 方法进行初始化工作。也提供了 beforeExecuteRoute($dispatcher)和 afterExecuteRoute($dispatcher)等相应的 hook 进行在处理请求之前及之后的操作接口。

1.5library 布局加载

该框架的布局一般如下所示：

```
app/
    config/
    controllers/
    library/
    models/
    plugins/
    views/
public/
    bootstrap/
    css/
    js/
```

所以 library 下的类一般会在开始的时间注册相应的目录，即可在程序中自动查找并加载相应的 lib 类。

1.6 模板

Phalcon 包含一个强大的和快速的模板引擎，它被叫做叫 Volt。默认请求下直接用 php 做为模板引擎，同时还可以引入并自定义模板引擎，比如可以用 smarty 等。

如果要引入其他模板引擎则必须继承\Phalcon\Mvc\View\Engine 类，可以重写__construct()及 render()方法等处理页面渲染逻辑，同时用 Phalcon\Mvc\View::registerEngines() 接受一个包含定义模板引擎数据的数组。每个引擎的键名是一个区别于其他引擎的拓展名。模板文件和特定的引擎关联必须有这些扩展名。如：

```
$view = new \Phalcon\Mvc\View();
//A trailing directory separator is required
$view->setViewsDir('../app/views/');
$view->registerEngines(array(
      ".my-html" => 'MyTemplateAdapter'
));
```

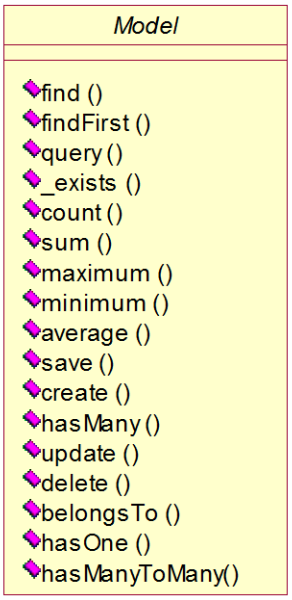
1.7ORM

模型代表了应用程序中的信息（数据）和处理数据的规则。模型主要用于管理与相应数据库表进行交互的规则。 大多数情况中，在应用程序中，数据库中每个表将对应一个模型。应用程序中的大部分业务逻辑都将集中在模型里，它是 ORM（对象关系映射）一种实现。

Phalcon\Mvc\Model 是 Phalcon 应用程序中所有模型的基类。它保证了数据库的独立性，基本的 CURD 操作， 高级的查询功能，多表关联等功能。 Phalcon\Mvc\Model 不需要直接使用 SQL 语句，因为它的转换方法，会动态的调用相应的数据库引擎进行处理。

模型是一个继承自 Phalcon\Mvc\Model 的一个类。 它必须放到 models 文件夹。一个模型文件必须包含一个类， 同时它的类名必须符合驼峰命名法。

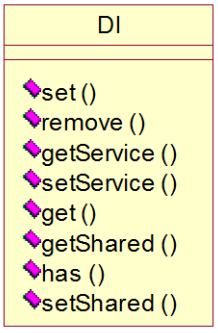
1.7.1Model 类图



从类图可以看出，model 抽象类提供了基本的 CURD 操作，一个 model 对应一个表，即这些操作是针对一个表的操作，内嵌已实现的 sql 方法，同时还提供了 hasOne、hasMany、hasManyToMany 等一对一，一对多，多对多等关系映射方法，也集成了 max、min 及 average 等计算操作，非常方便使用。

1.8IOC

Phalcon\DI 是一个实现依赖注入和定位服务的组件，而且它本身就是一个装载它们的容器。基本上，这个组件实现了控制反转的模式。类图如下所示：



如图所示，

remove (string $name)方法删除容器里的一个服务；

set (string $name, mixed $definition, [boolean $shared])该方法是在容器里注册一个服务，可以指定是否以共享单例的模式存在；

getService (string $name)获取容器里指定的服务；

setService (Phalcon\DI\ServiceInterface $rawDefinition)在容器里添加一下服务；

get (string $name, [array $parameters])获取容器里指定服务；

getShared (string $name, [array $parameters])以单例模式获取容器里的一个服务；
setShared (string $name, mixed $definition)以单例模式注册一个服务；
has (string $name)检查容器里是否有指定的服务；

2.Laravel

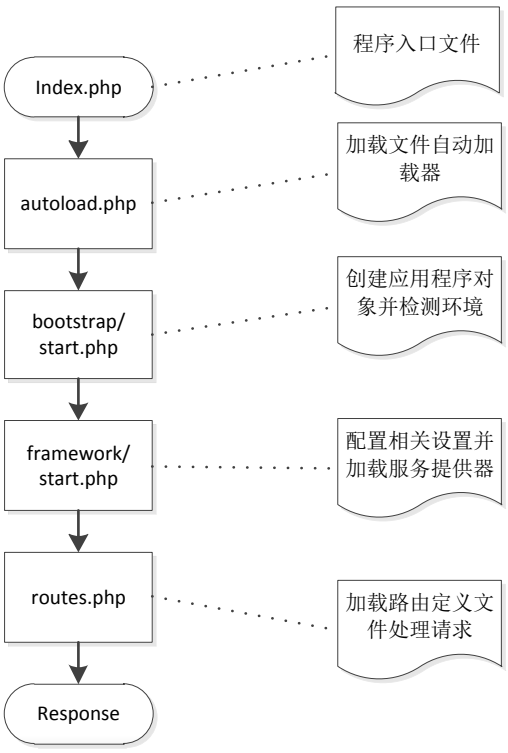
2.1 概述

Laravel 是一套 web 应用开发框架，它具有富于表达性且简洁的语法。是基于 Ruby on Rails、ASP.NET MVC、和 Sinatra 等开发语言或工具的。

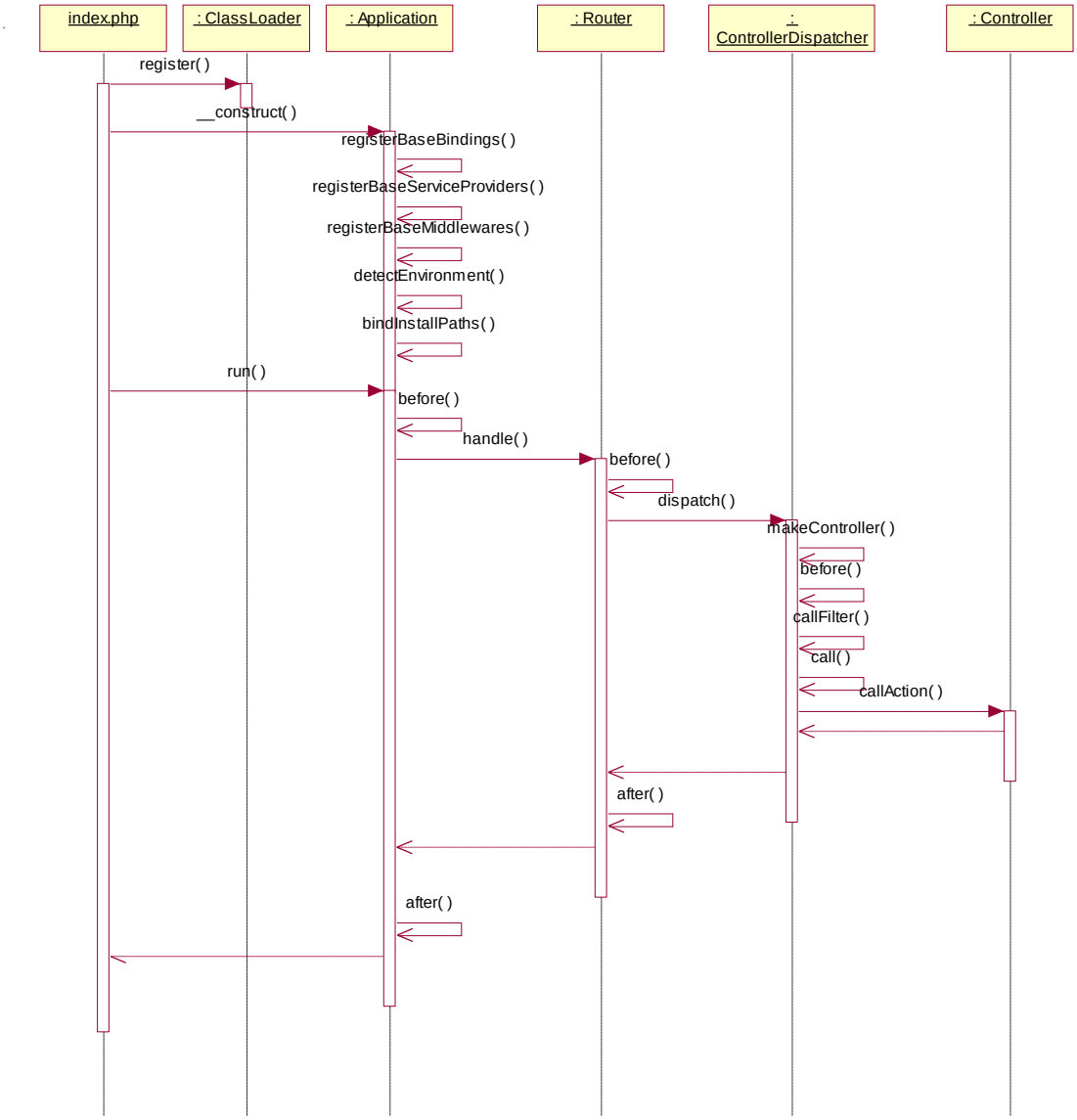
2.2 路由机制

该框架运行是依次 include 各个初始化文件，去初始化相应的模块数据。基本上路由信息都会在 app/routes.php 里进行配置定义，而具体的路由操作则由 Router 根据 Route 里的定义进行 handle 操作，然后会由 ControllerDispatcher 进行循环分发 dipatch 到各个具体的 Controller 进行相应的 action 操作。在这个流程中还提供了 App、Router、Controller 等相应的 before 和 after 等 hook，可以在各个流程之前及之后进行相应的操作。

2.2.1Laravel 框架运行流程图

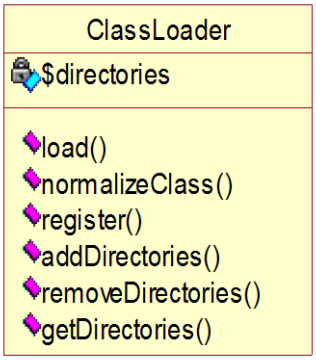


2.2.2 路由基本流程图



2.3 自动加载机制

Laravel 框架的自动加载机制是由 ClassLoader 类来实现，根据应用的配置文件目录，会在这些目录中去查找相应的类，并 require_once 进来，也是调用 php 原生的 spl_autoload_register 函数注册自己的 load 方法来实现的，类图所下所示：

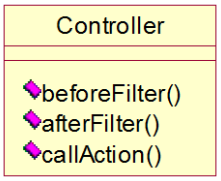


由图可以可以通过 addDirectories 方法来自行加入目录，这些目录下的类即可自动加载。在应用初始化的时候会自动加载相应的目录，如：

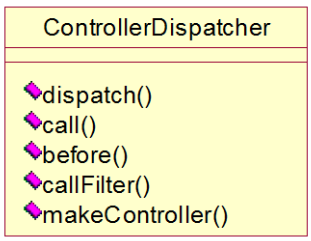
```
ClassLoader::addDirectories(array(

    app_path().'/commands',
    app_path().'/controllers',
    app_path().'/models',
    app_path().'/database/seeds',

));
```

2.4controller

所有的 controller 都要继承基类 Controller，该控制器有前后两个 hook，beforeFilter 和 afterFilter 分别可以在执行相应的动作之前或之后做一些操作。类图如下所示：



而所有的 controller 都由 ControllerDispatcher 来控制分发请求，创建相应的 Controller 和执行相应的 action 等，其类图如下所示：



定义如下所示：

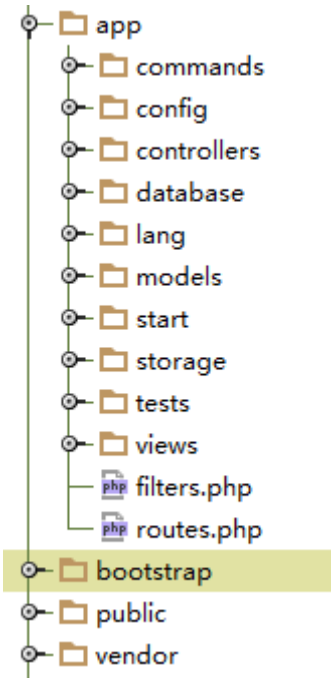
```
class UserController extends Controller {

    public function addUser()
    {
```

```
    }
    public function updateUser()
    {

    }

}
```

其中的 action 并不需要以 Action 用作后缀，直接在路由设置中把相应的 URI 对应到相应的 xxxController@yyyyy 即可。

2.5library 布局加载

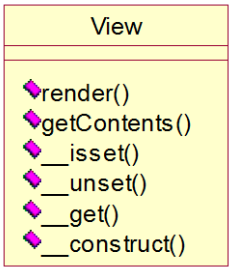
整个应用的一般布局如下图所示：



其中 vendor 是框架用到的所有的 lib，在初始化的时候会把相应的目录及其一些服务进行加载并初始化。由 composer 管理器下载生成所需的 lib

2.6 模板

模板用的是原生的 php 进行渲染，同时封装了 View 等类辅助视图操作，其类图如下所示：



也可以使用其他模板引擎，需要实现 EngineInterface 接口，并把实现的模板引擎传入 View 实例中使用。

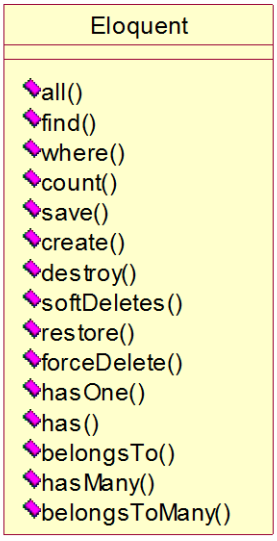
同时 Laravel 框架下的一个简单但又强大的模板引擎 Blade。不同于控制器布局，Blade 模板引擎由　模板继承和模板片段驱动。所有的 Blade 模板文件必须使用. blade.php　文件扩展名。

2.7ORM

Laravel 的 ORM 是由 Eloquent 实现的，所有的模型要继承 Eloquent，即可与数据库进行完善的交互。每个数据库表会和一个对应的「模型」互动。如果没有特别指定所对应的表，系统会默认自动对应名称为「类名称的小写复数形态」的数据库表。如下所示：

```
class User extends Eloquent {

    protected $table = 'my_users';

}
```

其类图如下所示：

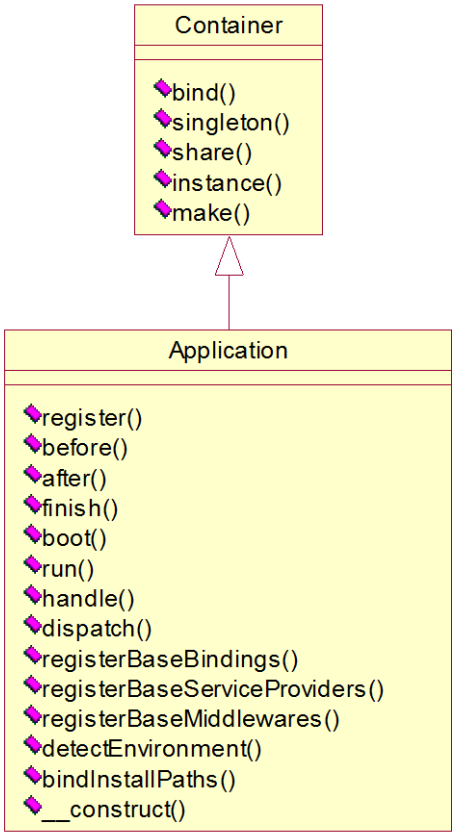


由图可知，Laravel 框架的 ORM 功能提供了数据库的 CURD 功能及聚合查询等，同时还提供了 hasOne,hasMany 和 belongsToMany 等一对一，一对多，多对多等关联查询功能。

2.8IOC

Laravel 使用 IoC（Inversion of Control，控制倒转，这是一个设计模式，可以先查看下百科）容器这个强有力的工具管理类依赖。依赖注入（也是一种设计模式，一般用于实现 IoC）是一个不用编写固定代码来处理类之间依赖的方法，相反的，这些依赖是在运行时注入的，这样允许处理依赖时具有更大的灵活性。

该框架的 IOC 容器是由 Application 来操作的，但其继承于 Container，所以实际是调用 Container 中的方法去实现服务的注入及获取，其类图如下所示：



其中：

bind()方法绑定一个类型到容器；

Make()方法是从容器中获得一个类型；

Singleton()方法是绑定一个单例模式的类型到容器，可以实现共享；

Instance()方法是绑定一个已经存在的一个对象到容器等。

3.与 ODP 相比

3.1 路由机制

两种框架虽然路由实现不同，但原理与 ODP 是一样的，都是根据 URI 进行映射匹配路

由到相应的 Controller，不同的是：

1、 Lavarel 框架 Controller 是一个类，相应的方法是 action，且 action 的命名没有什么限制，最终是有配置的路由指定到具体的 Controller@action。

2、 Phalcon 框架 Controller 同样是一个类，其中的方法是 action，但 action 的命名必须以 Action 为后缀，如 xxxAction 等，可以根据 URI 规则指定到相应的 XxxController@YyyAction 等。

3、 而 ODP 框架中的 Controller 一般根据 URI 用一个名字映射到一个单独的 Action，各人感觉前两种框架的 Controller 的规范更符合常理逻辑，一个 Controller 本身应该对应到一个业务模型，而不同的操作应该是 Controller 本身的一个操作方法。不过 ODP 也提供了简单的这个路由请求分发方法。

3.2 自动加载机制

这两种框架的类自动加载机制与 ODP 基本一致，无非是用 require_once,include 的区别，都可以注入配置一些目录及命名空间去查找并加载相应的文件类等。但其两个框架都可以更灵活的指定如 Controller、Model 等相应的目录，而 ODP 是根据事先约定好的目录，根据类名去查找，相对来说没有那么灵活。

其次，Lavarel 和 Phalcon 两个框架都会用到引用命名空间等，这个依赖于 php5.3.0 及以上版本，而 ODP 一般用的还是 php5.2.xx 版本，php5.3.xx 的版本用的还不是很多，这个 php 的版本感觉可以升级下，毕竟高版本会比低版本更好包括性能方面等。

3.3ORM

Lavarel 和 Phalcon 框架都有相应的 ORM 操作类，其原理基本上是一样的，一个 model 对应一个数据表，封装了对应表的 CURD 及聚合操作，同时还提供了关联操作等，这是借鉴了 Spring 框架及 Asp.net 等的设计思想。而 ODP 根本没有这些，只提供了数据库 sql 查询的封装而已。

3.4 总结

总上所述，我们可以从以下几点借鉴其设计思想。

3.4.1 路由及 Controller 的设计

Php 框架的路由思想基本上差不多，都是根据 URI 匹配来找相应的 Controller，甚至可以指定 URI 到一定的 Controller 上，这些与 ODP 差不多；唯一不同的是其两个框架都是把 Controller 做为请求分发器，同时通过自己相应的方法做为 Action 去执行相应的相应的操作，如 XxxController@YyyAction 等，所有的操作都集成在一个 Controller 类中，这个可以借鉴一下其设计思想，ODP 也提供了这些简单的操作，但用的很少，建议不是特别复杂的 action 逻辑都可以用这种方法来实现，如：

```
class IndexController extends Ap_Controller_Abstract {
        public function indexAction($name, $value) {
        }
```

```
    }
```

。

3.4.2IOC 依赖注入的设计

这两个框架在 IOC 上做了一些工作，都会在框架初始化的时候注入并初始化一些相应的服务，同时也可以在运行过程中显式的注入一些服务实例，比如 Session、Auth 等，还提供相应的过滤器做为 Hook 去实现面向切面的编程。ODP 其实也实现了一些简单的注入，在初始化的时候加载一些相应的文件类去初始化相应的运行环境，也提供了相应的 Hook，如 routerStartup、routerShutdown、preDispatch、postDispatch 等，但这些用户 plugin 好像是针对全局的，可以借鉴这两个种框架的 IOC 设计思想提供更多的依赖注入功能，比如在请求分发 Controller 前后等，不同的 Controller 执行动作 Action 前后都可以做一些相应的不同的操作；同时可以提供一个全局的容器 Container，可以在任何地方注册共用一些服务。

3.4.3ORM(对象关系映射)的设计

以上两个框架其实都封装了自己的 Model 类，该负责所有业务模块的逻辑操作，与数据库中的实体表一一对应，提供了方便的 CURD 及聚合操作。当然 ODP 主要更符合我们自己的搜索业务，与数据库交互的相对少一些，但我们可以借鉴其面向对象的设计思想，尽可能的抽取出相应的 Business Model 进行封装，模块化从而达到对外界透明的操作。

3.4.4 框架代码布局的设计

我们的 ODP 框架的代码其 app 和模板一般是分开存放，分别 check 下来，编译到相应的环境中运行。而以上两种框架其实是做为一个应用存在，模板和 app 在一个应用中，分别放在相应的目录中，这样可以直接把整个应用代码 check 下来，直接在环境中运行，实时修改不需要分别编译到相应的目录，可以很方便的开发，同时也可以自行指定相应的目录，如自行指定 Controller、Model 等相应的存在目录。如下所示：

```
Demo/
    app/
        app/controllers/
        app/library/
        app/models/
        app/views/
    public/
        index.php
        css/
        js/
```

我们可以借鉴这点设计，直接放在 webroot 下面开发，提高开发效率。

3.4.5 三个框架的优劣

1、性能上，ODP 和 Phalcon 框架都是用 C 写的 php 扩展，性能要比 Lavarel 框架好些。

2、功能上，Lavarel 封装了更多的方便可用的工具，而 Lavarel 和 Phalcon 框架在整体上与 ODP 相比，各模块耦合性更低，操作配置更灵活些。同时依赖的 PHP 版本比 ODP 要高，至少在 php5.3.0 及以上。

3、学习成本上，ODP 中的 AP 框架和 Phalcon 框架可直接把编译好.so 文件添加到 php

扩展，根据 API 文档即可上手开发；而 Lavarel 框架环境部署比较麻烦，需要安装各种依赖，但如果一旦建好，可用 composer 来管理依赖类库，上手还是比较容易的，同时可以查看 php 源码进行深入学习。不过个人还是用 ODP，因为对 ODP 比较熟悉，可以在 AP 框架的基础上借鉴其他框架的好的设计思想运用的项目开发中。

4、代码布局上，Phalcon 和 Lavarel 框架代码按整个应用存放，目录分明，在环境中结构清晰可见，直接 check 开发管理等，而 ODP 目录虽然分明，但代码分开存放，需要分别编译到相应的环境中开发调试，相对来说麻烦点。

总体上大部分 php 框架都大同小异，无非是多封装了一些工具类，提供了更方便的操作。ODP 框架唯一缺少的主要是 ORM 这一块，不过可以尝试在 AP 框架的基础上用 php 去实现，同时需要学习 webserver 的一些知识，如 nginx, apache 等配合学习 PHP 及框架会更好。以上是我的一些略见，仅供参考。