一、composer 介绍

a) Laravel 支持组件化开发

b) TinkPHP 属于功能性框架

c) 是 PHP 用来管理依赖（dependency）关系的工具。你可以在自己的项目中声明所依赖的外部工具库（libraries），Composer 会帮你安装这些依赖的库文件。

二、composer 的安装

a) wget https://getcomposer.org/composer.phar 建议在家目录下载

b) chmod +x composer.phar 添加可执行权限

c) composer 脚本拷贝一份到环境变量目录下(为了执行方便)

i. 查看环境变量 echo $PATH

ii. cp -a composer.phar /usr/bin/composer

三、lavavel 安装

a) 全局安装

i. composer global require "laravel/installer" （全套 laravel）

ii. 例如：laravel new blog 命令会在当前目录下建立一个名为 blog 的目录，此目录里面存放着全新安装的 Laravel 并安装所有依赖包。

iii. 全局安装后 laravel 可执行文件在~/.composer/vendor/bin 目录下 ,需要配置 PATH 环境变量， 这样可执行文件 laravel 就能被你的系统检测到。

b) 局部安装

i. composer create-project laravel/laravel --prefer-dist 安装一个 laravel 应用，并在当前执行目录下安装。（需要等待一些时间，建议在网速好情况下安装，免得出现的编辑文件时奇怪的现象）

四、本地域名解析与 apache 虚拟主机配置

a) 设置域名映射同一个 IP ,修改 hosts :C:\Windows\System32\drivers\etc\hosts

192.168.1.223 www.cc.com

b) 配置虚拟主机

```
<VirtualHost *:80>
    ServerAdmin webmaster@dummy-host23.example.com
    #DocumentRoot "/Web/apps/apache2/htdocs//cc"
    DocumentRoot "/Web/apps/apache2/htdocs//cc/laravel/public"
    ServerName cc.com
    ServerAlias www.cc.com
        <Directory />
                Options FollowSymLinks
                DirectoryIndex index.php index.html
                AllowOverride All
                Order deny,allow
                Allow from all
        </Directory>
    ErrorLog "logs/dummy-host.example.com-error_log"
    CustomLog "logs/dummy-host.example.com-access_log" common
</VirtualHost>
```

c) 目录权限

i. web 服务器需要拥有 storage 目录下的所有目录和 bootstrap/cache 目录的写权限 （网站缓存文件目录 chmod -R 777 storage ）

ii. 也可以修改用户组 chown -R apache:apache storage

五、wike 的下载与安装

a) 一个文档系统，用来构建个人知识库，可定制性强

b) 官网 https://www.mediawiki.org/wiki/Download/zh

c) 下载 wget https://releases.wikimedia.org/mediawiki/1.23/mediawiki-1.23.13.tar.gz

d) tar -zvxf mediawiki-1.23.13.tar.gz

e) mv mediawiki-1.23.13 /Web/apps/apache2/htdocs/wiki

f) 直接访问对应虚拟主机即可安装（根据每一步提示安装）

六、git 仓库的建立与 git 服务器的搭建

a) 查看 git 版本 git --version

b) 编译安装（http://www.cppblog.com/Error/archive/2014/01/14/205365.aspx）

c) .gitignore 里面内容是文件的名称，表示这些文件都不需要使用 git 管理（laravel 默认版本管理是 git，所以在 laravel 目录会这个文件存在）

d) 项目使用 git 版本管理

i. 初始化 git，在项目的根目录下（/Web/apps/apache2/htdocs/cc/laravel）

[root@localhost laravel]# git init

Initialized empty Git repository in /Web/apps/apache2/htdocs/cc/laravel/.git/

备注 :此时在项目的根目录下有生成一个 .git 的文件夹（存放配置和版本的记录信息）

e) Git 基本命令（建议在网站根目录下操作）

i. 查看 git 版本状态 git status

ii. 添加文件到版本库暂存区里面 git add 文件名 [--all 添加所有文件](svn)

iii. 提交文件到版本库 git commit -m “说明本次提交”

iv.

f) 在服务器上部署 Git

i. 创建 git 用户，用来管理 git adduser git

ii. 创建 git 裸仓库 git clone -- bare 项目[demo] 自定义目录[demo.git]

git 裸仓库表示在项目中/Web/apps/apache2/htdocs/cc/laravel/.git 目录下的文件，目录包含项目文件，在开发我们作为项目源提供给开发者协同开发

```
[root@localhost cc]# ll
drwxr-xr-x 12 root root       4096 3 月    23 23:31 laravel
drwxr-xr-x  7 root root       4096 3 月    23 23:31 laravel.git
```

iii. 修改裸仓库的所属组 chown -R git:git laravel.git

iv. 把克隆项目移动到 git 家目录下 mv laravel.git /home/git/

v. 本地克隆 git 项目 git clone git@192.168.1.223: laravel.git

vi. 在 mac 本地创建密钥文件 ssh-keygen -t rsa 生成两个密钥文件

id_rsa 密钥 id_rsa.pib 公钥 把公钥上传到服务器（直接 clone 无需密码）

1.在 git 家目录 mkdir .ssh 2.cat id_rsa.pib >> .ssh/authorized_keys

2.将复制得到的公钥添加 git 家目录/.ssh/authorized_keys，如果没有该文件手动创建，修改权限：chmod 600 authorized_keys。

七、laravelDebug 安装与调试命令（多下载几次）

a) 进入项目根目录下输入 composer require barryvdh/laravel-debugbar

i. 在项目根目录下 config / app.php 中服务器提供者列表中添加 Barryvdh\Debugbar\ServiceProvider::class 在别名配置数组中添加 'Debugbar'=>Barryvdh\Debugbar\Facade::class

ii. 在项目根目录执行 php artisan vendor::publish

iii. 在/laravel/vendor/Barryvdh 对应的第三方扩展

b) 设置 git

i. 配置用户 git config --global user.name “yuncopy”

ii. 配置邮箱 git config --global user.email “yuncopy@sina.com”

iii. 项目目录下添加所有文件 git add .

iv. 查看状态 git status

v. 提交文件到缓存区 git commit -m “debug init”

vi. 检出一个分支 git checkout -b “描述”

vii. 查看当前分支 git branch

viii. 切换主分支 git checkout master

ix. 隐藏 index.php 需要开启 apache 重写模块

x. 开启 AllowOverride all

c) 格式化调试函数

i. dd(需要打印的数组)

d) 修改数据库

i. .env 文件

八、环境配置与数据库连接

a) 配置文件目录 config

i. 数据库配置文件 database.php

ii. 数据环境变量文件 .env

iii. 查看 git 版本控制器忽略的文件 .gitignore

九、laravel 数据迁移工具

a) 为数据库迁移就像版本控制,允许一个团队很容易修改和共享应用程序的数据库模式，迁移通常是搭配 Laravel 的模式构建器轻松地构建应用程序的数据库模式。

b) 创建数据表文件 php artisan make:migration create_users_table （表名）

c) 迁移出你所有创建的数据库 php artisan migrate （检查是否已存在的表）

d) 数据库创建的目录 /laravel/database/migrations

e) 详细请阅读官方文档 |-数据库 |--迁移 即可（自带有实例）

十、laravel 静态资源管理 Elixir 工具

a) 编译 less 和 sass ，合拼 CSS 和 JS 减少资源的请求提供访问速度

b) 我们习惯 js 文件按照功能模块区分，方便维护，但是请求次数增多。

c) node_modules 是 node.js 的组件目录 ，package.json 是 node.js 组件的配置文件，vendor 是 php 的组件目录，composer.json 是 php 组件的配置文件。

d) 安装部署 node.js

i. 下载 （发行版）

wget https://nodejs.org/dist/latest-v4.x/node-v4.4.2-linux-x64.tar.gz

wget https://nodejs.org/dist/v4.4.2/node-v4.4.2.tar.gz （源码版）

ii. 解压配置安装 （试用源码版）

1. yum -y install g++
2. tar -zxvf node-v4.4.2.tar.gz
3. cd node-v4.4.2-linux-x64
4. ./configure --prefix=/Web/apps/node
5. make && make install
6. 配置 NODE_HOME，进入 profile 编辑环境变量

a) vim /etc/profile

```
#set for nodejs
export NODE_HOME=/Web/apps/node
export PATH=$NODE_HOME/bin:$PATH
```

b) :wq 保存并退出，编译/etc/profile 使配置生效

```
source /etc/profile
```

7. 验证是否安装配置成功

a) node -v

8. npm 模块安装路径

a) /Web/apps/node/lib/node_modules/

b) 注：Nodejs 官网提供了编译好的 Linux 二进制包，下载直接应用。

iii. 解压配置安装 （试用发行版）二进制包

1. 查看系统位数 getconf LONG_BIT

2. 到官网下载对应的版本 （谨记：版本一定）

wget https://nodejs.org/dist/latest-v4.x/node-v4.4.2-linux-x64.tar.gz（64）

wget https://nodejs.org/dist/latest-v4.x/node-v4.4.2-linux-x86.tar.gz（32）

3. tar -zxvf node-v4.4.2-linux-x86.tar.gz

4. mv node-v4.4.2-linux-x64 /Web/apps/node

5. 添加对应的环境变量或者使用软连接到系统变量目录下

十一、laravel 静态资源管理 Elixir 工具使用

a) 在项目根目录下安装 gulp ： npm install --global gulp

b) 在项目根目录下安装依赖包： npm install（多试几次，保证网速）

c) 成功会在/laravel/node_modules 分别有 bootstrap-sass ,gulp ,laravel-elixir 目录（此文件夹目录大小很大）

d) 主要应用在合并资源，因为减少线程的请求

e) 使用（具体查看官方手册 https://laravel.com/docs/5.2/elixir）

i. CSS 资源路径 resources/assets/css （没有目录可以创建）

ii. 合并后文件路径 public/css/all.css

iii. vim /laravel/gulpfile.js 编辑

```
elixir(function(mix) {
    mix.styles([
        'normalize.css',
        'main.css'
    ], 'public/assets/css');
});  // 合并 normalize.css，main.css 到 public/assets/css 目录 all.css
```

iv. 在项目根目录下执行 gulp 即可 （会有提示信息报错不要管它）

v. CSS，JS，SASS ，LESS 同理，按照手册说明使用即可（官方文档地址：https://laravel.com/docs/5.2/elixir）

十二、其他类型的资源管理与 sass 编译

a) 创建对应的目录 resources/assets/js

```
elixir(function(mix) {
    mix.browserify('main.js');
});
```

生成 public/js/main.js

b) 合并 sass (resources/assets/sass)

app.sass 导入资源

import “node_modules/bootstrap-sass/assets/stylesheets/bootstrap”

c) 其他资源请查阅官方文档

十三、测试驱动开发

a) 测试单元 phpunit.xml 配置文件

b) 测试脚本

i. laravel\tests\TestCase.php

c) 测试案例

i. laravel\tests\ExampleTest.php 遵守控制器命名规则

d) 使用 命令行执行执行脚本 laravel\vendor\bin\phpunit

e) 开发思路：先开发测试驱动在开发项目

十四、附加-使用本地 IDE 连接到远程主机进行项目开发

a) 使用工具 PhpStorm

b) 配置密钥远程连接

i. 在 root 家目录执行 ssh-keygen -t rsa

ii. 回车确认并且进入目录 cd ./ssh 生成 d_rsa id_rsa.pub 公钥和私钥

iii. 重定向 cat id_rsa.pub >> authorized_keys

iv. 复制到/tmp 修改权限保证能直接下载私钥 id_rsa

v. 使用 sftp 连接远程服务器时选择密钥登录

十五、laravel 项目生命周期 （执行流程：一个请求发起和响应结束的过程）

a) 先执行应用的入口文件 public/index.php，包含 bootstrap/autoload.php 并自动加载框架中核心类和函数（IOS 容器），完成后 bootstrap/app.php 创建实例化应用。bootstrap 目录包含的几个文件用于启动框架和配置自动加载功能，还有一个 cache 目录，用于存放框架自动生成的文件，能够加速框架启动。

b) 接着传入的请求发送到 HTTP 内核或控制台的内核 app/Http/Kernel.php 并且继承 Illuminate\Foundation\Http\Kernel 使用 bootstrappers 数组定义在请求之前加载引导程序配置错误处理,配置日志记录，还定义 middleware 中间件是所有请求之前必须通过由应用程序来处理。读写操作由中间件处理 HTTP 会话,确定应用程序是否在维护模式,验证 CSRF 令牌等等。handle 方法是处理一个请求和响应

c) 加载服务提供者，最重要的一个内核引导行动加载应用程序的服务提供者。config/app.php 文件中以数组形式罗列，首先注册方法后在引导方法将被调用，服务提供商负责引导所有的框架的各种组件,如数据库、队列、验证和路由组件等。

d) 发起请求，请求将交给调度的路由器。路由器将请求调度路线或控制器,以及运行任何特定的中间件。

e) 使用 Facades 将创建类中的方法并合并成”静态“的方法，给对应的创建对象定义别名。对服务的提供者创建别名方便使用类中的方法。

f) HTTP（浏览器）/命令行

十六、应用程序结构

a) app 包含应用程序的核心代码

i. Console 命令行操作和编写存放目录

ii. Events 事件监听

iii. Jobs 处理消息，计划任务

b) Bootstrap 用于启动框架和配置自动加载功能，还有一个 cache 目录，用于存放框架自动生成的文件，能够加速框架启动。

c) config 包含所有应用程序的配置文件

d) database 目录包含了数据库迁移与数据填充文件

e) public 目录包含前面的控制器和你的资源文件（图片、JavaScript、CSS，等等）。

f) resources 目录包含你的视图、原始的资源文件（LESS、SASS、CoffeeScript）和本地化语言文件。

g) storage 目录包含编译后的 Blade 模板、基于文件的 session、文件缓存和其他由框架生成的文件。

i. app 目录用户存放应用程序所用到的任何任何文件；

ii. framework 目录用于存放由框架生成的文件和缓存文件；

iii. logs 目录用于存放应用程序的日志文件。

h) tests 目录用于存放你的自动化测试文件。默认自带了一个 PHPUnit 的实例。

i) vendor 目录用于存放 Composer 的依赖包。

十七、HTTP 发起请求

a) 加载 HTTP 内核文件 laravel\app\Http\Kernel.php

b) 加载中间件 The application's route middleware groups 进行验证（全局中间件和单个路由中间件，中间件一般处理验证，数据过滤，权限验证等）

十八、模板使用-模板模块加载

a) 视图包含了应用程序渲染的 HTML 数据，并将应用程序的显示逻辑与控制逻辑有效的分离开，在 Laravel 中，视图被保存在 resources/views 目录中。

```
<!-- View stored in resources/views/greeting.php -->
<html>
        <body>
                <h1>Hello, <?php echo $name; ?></h1>
        </body>
</html>
```

b) 详细使用 http://www.golaravel.com/laravel/docs/5.1/views/

i. 子目录嵌套渲染方式

ii. 判断视图是否存在

iii. 视图数据

1. 视图模板传递数据

a) return view('greetings', ['name' => 'Victoria']);

b) $view = view('greeting')->with('name', 'Victoria');

2. 把数据共享给所有的视图

3. 视图组件

4. 视图创建者

c) Blade 模板

i. Blade 不局限使用原生 PHP 代码。所有 Blade 视图页面都将被编译成原生 PHP 代码并缓存起来，除非模板文件被修改了，否则不会重新编译，减轻额外负担。Blade 视图文件使用 .blade.php 文件扩展名，并且一般被存放在

resources/views 目录。

ii. 使用 Blade 能获得的两个主要好处是 模板继承(template inheritance ）和视图片断（ sections ）

1. 引入子视图

   a) @include 指令允许你方便地在一个视图中引入另一个视图

2. 继承

   a) laravel\resources\views\child.blade.php

```
@extends('layouts.master'){{--继承--}}
@section('title', 'Page Title'){{--实现父类中 @yield('title')--}}
@section('sidebar')
    @@parent {{--继用父类--}}
    <p>This is appended to the master sidebar.</p> {{--并追加--}}
@endsection
{{--实现父类中 @yield('title')--}}
@section('content')
    <p>This is my body content.</p>
@endsection
```

   b) laravel\resources\views\layouts\master.blade.php

```
<html>
<head>
    <title>App Name - @yield('title')</title>
</head>
<body>
@section('sidebar')
    This is the master sidebar.
@show
<div class="container">
    @yield('content')
</div>
</body>
</html>
```

3. 模板详细用法 http://www.golaravel.com/laravel/docs/5.1/blade/

4. 展示数据，使用 PHP 函数

5. 页面中使用逻辑判断，流程控制

十九、控制器与 resfulApi 的使用

a) 控制器类一般存放在 app/Http/Controllers 目录下

b) 查看 Laravel 版本 php artisan --version （在项目根目录下执行）

c) 只有 5.1 版本支持 （php artisan make:controller PhotoController）

i. composer create-project laravel/laravel=5.1.* --prefer-dist

ii. 备注：5.2 版本执行创建控制器时无法创建对应的 restfulApi 方法

d) 创建基本控制器类

i. 路由

```
Route::get('user/{id}', 'UserController@showProfile');
```

ii. 控制器

```
php artisan make:controller PhotoController
/**
* 显示指定用户的个人信息
 *
* @param    int    $id
* @return Response
*/
public function showProfile($id)
{
    return view('user.profile', ['user' =>$id]);
}
```

iii. 视图

```
laravel\resources\views\user\profile.blade.php
Hello, {{ $user }}
```

e) 控制器 & 命名空间

i. 创建控制 （项目根目录）

1. [root@localhost laravel]# php artisan make:controller Photo/AdminController

2. 命名控制器路由 （取别名）

a) Route::get('foo', ['uses' => 'FooController@method', 'as' => 'name']);

b) $url = route('name'); http://192.168.1.223/foo

c) $url = action('UserController@method');

f) RESTful 资源控制器

i. 资源管理器处理的动作

Verb	Path	Action	Route Name
GET	/photo	index	photo.index
GET	/photo/create	create	photo.create
POST	/photo	store	photo.store
GET	/photo/{photo}	show	photo.show
GET	/photo/{photo}/edit	edit	photo.edit
PUT/PATCH	/photo/{photo}	update	photo.update
DELETE	/photo/{photo}	destroy	photo.destroy

ii. 局部资源路由

1. 指定控制器中有效的方法

Route::resource('photo', 'PhotoController',

['only' => ['index', 'show']]);

2. 排除控制器中的方法

Route::resource('photo', 'PhotoController',

['except' => ['create', 'store', 'update', 'destroy']]);

iii. 命名资源路由（还没有理解）

iv. 更多详细见官网手册

二十、服务器容器与容器与工厂模式

a) 核心框架对象 laravel\bootstrap\app.php

i. Application extends Container （继承容器对象）

ii. $app = require_once __DIR__.'/../bootstrap/app.php'; //返回 app 对象

iii. 服务器容器绑定

1. 原始使用对象中方法

laravel\app\Http\routes.php

```
Route::get('apple', function () {
    $this->app->bind('A',function($app){
        return new \App\User();
    });
    $apple = $this->make('A');
    dd($apple);
});
```

2. 业务逻辑代码不改变，只改变对象

laravel\bootstrap\app.php

```
$app->bind('A',function(){
    return new App\Http\Controllers\PhotoController();
    //return new App\User();   //改变对象，对 make 方法没有任何影响
});

Route::get('apple', function () {
    $apple = $this->make('A'); //这里不做任何修改，只传递对象名 A 集合
    dd($apple);
});
```

3. 生活中实例

Make 类似工厂，只要给一个图纸 A 就能生产一对象

```
$apple = $this->make('A');
```

Bind 类似一张图纸，只要图纸编号不变，送到工厂生产

```
$app->bind('A',function(){
    return new App\Http\Controllers\PhotoController();
});
```

二十一、服务器容器单列与实例绑定

a) 单例绑定对象，避免相同对象冲突

i. laravel\bootstrap\app.php

```
$app->singleton('A',function(){
    return new App\Http\Controllers\PhotoController();
});
```

ii. 获取对象

1. $fooBar = $this->app['FooBar'];

2. $fooBar = $this->app->make('FooBar');

b) 更多详细见官网手册

二十二、依赖注入（类之间使用）--Binding Interfaces To Implementations

a) laravel\bootstrap\app.php

i. $this->app->bind('\App\EventB', 'App\EventB');
ii. $this->app->bind('\App\EventA', 'App\EventA');

b) laravel\bootstrap\app\EventB.php

```
namespace App;
class EventB
{
    private $ta;
    private function _construct(\App\EventA $a){
        $this->$ta = $a;
    }
}
```

c) laravel\bootstrap\app\EventA.php

```
namespace App;
class EventB
{
    private $tb = 123;
}
```

d) laravel\app\Http\routes.php

```
Route::get(tba, function () {
    $apple = $this->make(\APP\TB::class);
    dd($apple);
});
```

e) 服务器容器能够将一个接口绑定到特定的实现，在一个构造函数时需要传递对象时

只要在服务器容易绑定即可实例化一个对象，而不是人工 new 对象再传参数，而是使用容器管理对象。

f) 其实依赖注入就是 A 类中的某一个方法要 B 类以参数的形式传入到其方法中。

二十三、服务提供者与 laravel 低耦合架构原理

a) 由于合作开发，减少多个人操作同一个文件，导致合并代码时发生冲突。

b) laravel\vendor\laravel\framework\src\Illuminate\Foundation\Http\Kernel.php

i. protected $bootstrappers = [ ]; //遍历循环初始化类

ii. 框架中的每个功能就是每个系统中提供的系统服务，使用容器管理

c) 注册自定义类（根据系统案例：一定要添加在服务提供数组列表中）

i. laravel\app\Providers\AppServiceProvider.php // 服务提供者

```
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        $this->app->bind('\App\EventB', '\App\EventB');
        $this->app->bind('\App\EventA', '\App\EventA');
    }
}
```

d) 服务提供者就是用来扩展框架功能的，注册绑定即可

e) laravel\config\app.php 自定义服务提供者时一定要加入服务提供者 providers 配置中，同样再加入新的第三方服务时也需要此操作（如：添加 DUG 调试）

```
laravel\config\app.php
'providers' => [ ]
```

二十四、Facades 的使用

a) 一般我们使用类

//首先在服务器提供者中单例绑定类

```
public function register()
{
        // $this->app->bind('\App\EventB', '\App\EventB');
        // $this->app->bind('\App\EventB', '\App\EventB');
         $this->app->singleton('\App\EventA', '\App\EventA'); //singleton 单例
         $this->app->singleton('\App\EventA', '\App\EventA');
  }
```

//使用时先实例化 make 才能使用类中的方法

```
Route::get('apple', function () {
    $ta = $this->app->make(\App\TA::class);
    $ta->title ='aaa';
    echo $ta->title;
});
```

b) Facades 优势

i. $users = **DB::**table('users')->count(); //使用时类似于静态访问其中的方法

c) 自定静态化接口

\laravel\app\TB.php

```
<?php
namespace App;
class TB
{
    public   $tb;
    public   $title;
    public function __construct(\App\TA $a)
    {
        $this->tb=$a;
    }
    public   function   setTitle($item)
    {
        $this->title = $item;
    }
    public   function   getTitle()
    {
```

```
            return $this->title;
        }
}
```

\laravel\app\Http\Z.php

```
namespace    App\Http;
class    Z extends    \Illuminate\Support\Facades\Facade
{
        /**
         * Get the registered name of the component.
         *
         * @return string
         */
        protected static function getFacadeAccessor()
        {
                return '\App\TB';
        }
}
```

d) 使用静态化方法

\laravel\app\Http\routes.php

```
Route::get('cache', function () {
        \App\Http\Z::setTitle('1234');
        echo    \App\Http\Z::getTitle();
});
```

e) 注册到别名

\laravel\config\app.php

```
'aliases' => [
        'Z'                 => \App\Http\Z::class,
]
```

直接使用

```
Z::setTitle('asdf');
echo Z::getTitle();
```

二十五、中间件的用途

a) 问题

i. 数据库灌水

ii. 跨域请求

iii. 验证用户登录

b) 每个功能负责各自任务

c) 请求（验证）/响应（设置头信息等）

d) 中间件功能

二十六、中间件的创建和使用

a) 定义

i. HTTP 中间件为过滤访问你的应用的 HTTP 请求提供了一个方便的机制.

b) 创建中间件（在项目根目录下）

i. php artisan make:middleware OldMiddleware

c) 框架中生命周期

i. public/index.php（入口文件）-->bootstrap/app.php（初始化加载类）

-->Http\Kernel.php/Console\Kernel.php（核心类库）-->

+ protected $middleware = [ ] 全局中间件

+protected $routeMiddleware = [ ] 局部中间件（特定路由）

-->Middleware（注册中间件）-->Http\routes.php（路由）负责访问指定

路径会转交给绑定的控制器处理 -->中间件-->控制器

d) 中间件必有函数

```
public function handle($request, Closure $next)
{
    return $next($request);
    //中间处理完成后移交给路由的第二个参数处理（闭包函数或者控制器）
}
```

e) 配置自定义中间件

i. 全局中间件

1. 如果你期望中间件在每一个 HTTP 请求时都会执行，只需要将中间件类加入到 app/Http/Kernel.php 类的 $middleware 属性的列表中。

```
protected $middleware = [
    \App\Http\Middleware\OldMiddleware::class,
];
```

ii. 指派中间件给路由

1. 首先在 app/Http/Kernel.php 文件中指定一个键值。默认情况下，这个类的 $routeMiddleware 属性。

```
protected $routeMiddleware = [
    'old' => \App\Http\Middleware\OldMiddleware::class,
];
```

2. 路由选项数组中使用 middleware 键来指派

```
Route::get('/profile', ['middleware' => old, function () {

}]);
```

3. 使用 middleware 方法指定中间件

```
Route::get('/', function () {
    //
}])->middleware(['first', 'second']);
```

iii. 中间件也可以接收额外的自定义参数

二十七、LaravelRequest

a) 获取请求数据

```
public function store(Request $request)
{
    $name = $request->input('name');
    //
}
```

b) 避免重复提交，必须携带 Token 进行验证

c) 打印 Request 对象封装大量有关于请求的信息 dd（$request）

d) 具体基本用法请查阅官方手册

二十八、控制器数据验证模块

a) 表单提交数据进行验证（http://www.golaravel.com/laravel/docs/5.1/validation/）

b) 使用方法

```
/**
 * Store a new blog post.
 *
 * @param    Request    $request
 * @return Response
 */
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'name' => 'required',
        email=> 'required|email',
    ]);
    // The blog post is valid, store in database...    执行后面的代码
}
```

c) 错误信息提示

```
<h1>Create Post</h1>

@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

d) 验证与控制器分离，减低耦合

i. 创建验证类

1. php artisan make:request StoreBlogPostRequest

ii. 定义规则使用

1. 生成类将被放置在 app / Http /请求目录 ,添加一些验证规则的 rules 方法:

```
/**
 * Get the validation rules that apply to the request.
 *
 * @return array
 */
public function rules()
{
    return [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ];
}
```

iii. 这意味着你不需要任何验证打乱您的控制器逻辑

```
/**
 * Store the incoming blog post.
 *
 * @param    StoreBlogPostRequest    $request
 * @return Response
 */
public function store(StoreBlogPostRequest $request){
    // The incoming request is valid...
}
```

e) 更多使用方法查阅官方手册(http://www.golaravel.com/laravel/docs/5.1/validation/)

二十九、防止跨站请求，启用 csrf 认证

a) VerifyCsrfToken （保证请求由我们服务器发起）

```
public function handle($request, Closure $next)
{
    if ($this->isReading($request) || $this->shouldPassThrough($request) ||
        $this->tokensMatch($request)) {
        return $this->addCookieToResponse($request, $next($request));
    }

    throw new TokenMismatchException;
}
```

//isReading 判断请求页面是否为渲染页面

//shouldPassThrough 判断请求页面是否为不需要验证的

// tokensMatch 判断请求页面是否匹配 Token

b) 获取 Token

```
protected function tokensMatch($request)
{
    $sessionToken = $request->session()->token();
    $token = $request->input('_token') ?: $request->header('X-CSRF-TOKEN');
    if (! $token && $header = $request->header('X-XSRF-TOKEN')) {
        $token = $this->encrypter->decrypt($header);
    }
    if (! is_string($sessionToken) || ! is_string($token)) {
        return false;
    }
    return Str::equals($sessionToken, $token);
}
```

c) 获取服务器生成 Token

```
<form action="/foo/bar" method="POST">
    <input type="hidden" name="_method" value="PUT">
    <input type="hidden" name="_token" value="{{ csrf_token() }}">
</form>
```

d) 路由很重要

http://www.golaravel.com/laravel/docs/5.1/routing/

三十、Laravel 表单助手

a) 下载安装（保持与 laravel 版本一致）

i. [root@localhost laravel]# composer require laravelcollective/html=5.1.*

b) 在 config/app.php 中配置服务提供者

i. Collective\Html\HtmlServiceProvider::class

c) 添加使用时别名

i. 'From' => Collective\Html\FormFacade::class

ii. 'Html' => Collective\Html\HtmlFacade::class

d) 视图页面中 PHP 代码原样输出 {!! phpcode }

e) 使用方式（生成 HTML 元素并进行提交验证同时保持表单原来数据）

```
{!! Form::open(array('route' => 'yuyue.store', 'class' => 'form',
    'novalidate' => 'novalidate')) !!}

    <div class="form-group">
        {!! Form::label('Your Name') !!}
        {!! Form::text('name', null,
            array('required',
                'class'=>'form-control',
                'placeholder'=>'Your name')) !!}
    </div>

    <div class="form-group">
        {!! Form::label('Your E-mail Address') !!}
        {!! Form::text('email', null,
            array('required',
                'class'=>'form-control',
                'placeholder'=>'Your e-mail address')) !!}
    </div>

    <div class="form-group">
        {!! Form::label('Your Message') !!}
        {!! Form::textarea('message', null,
            array('required',
                'class'=>'form-control',
                'placeholder'=>'Your message')) !!}
    </div>

    <div class="form-group">
        {!! Form::submit('Contact Us!',
          array('class'=>'btn btn-primary')) !!}
    </div>
    {!! Form::close() !!}
```

三十一、ORM 模型映射的建立与使用

a) 数据库操作使用

b) ORM 定义

i. Laravel 所自带的 Eloquent ORM 是一个优美、简洁的 ActiveRecord 实现，用来实现数据库操作。每个数据表都有一个与之相对应的“模型

（Model）”，用于和数据表交互。模型（model）帮助你在数据表中查询数据，以及向数据表内插入新的记录.

ii. 创建一个 Eloquent 模型（model），文件通常被放在 app 目录下。

1. php artisan make:model Flight

iii. 生成 model 同时生成数据库迁移，可添加 --migration 或 -m 参数来实现

1. php artisan make:model Flight --migration

iv. 执行命令生成数据库迁移文件(http://www.golaravel.com/laravel/docs/5.1/migrations/)

1. php artisan migrate //配置字段信息创建表

v. 使用方式

```
public function store(Request $request)
{
    // Validate the request...
    $flight = new Flight;
    $flight->name = $request->name;

    $flight->save();
}
```

c) 更多使用方式请查阅手册

三十二、ORM 映射利弊分析

a) 约束性强

b) 级联查询，更新，删除

三十三、利用邮件服务将用户表单推送到手机

a) 安装下载 HTTP 请求模块

i. composer require guzzlehttp/guzzle=6.*

ii. composer.json 的 require 一项中加入"guzzlehttp/guzzle": "~5.3|~6.0"

然后运行：composer update

b) 配置邮件信息

i. laravel\config\mail.php

'from' => ['address' => null, 'name' => null]

ii. laravel\.env

MAIL_DRIVER=smtp

MAIL_HOST=smtp.163.com

MAIL_PORT=25

MAIL_USERNAME=null

MAIL_PASSWORD=null

MAIL_ENCRYPTION=null

c) 使用方式

```
public function sendEmailReminder(Request $request, $id)
{
    $user = User::findOrFail($id);

    Mail::send('emails.reminder', ['user' => $user], function ($m) use ($user) {
        //$m->from('hello@app.com', 'Your Application');   //发送邮箱
        $m->to($user->email, $user->name)->subject('Your Reminder!');
    });

    //emails.reminder  邮件模板

    //function ($m) use ($user)    PHP5.3  闭包函数

}
```

三十四、利用 Laravel 认证模板完成用户注册登录

a) 访问控制（权限控制）

b) 使用步骤

i. 配置路由

```
Route::get('login', [
    'as' => 'login',
    'uses' => 'Auth\AuthController@getLogin'
]);

Route::get('logout', [
    'as' => 'logout',
    'uses' => 'Auth\AuthController@getLogout'
]);

Route::controllers([
    'auth' => 'Auth\AuthController',
    'password' => 'Auth\PasswordController',
]);
```

ii. 配置跳转路径

1. protected $redirectPath = '/dashboard';
2. protected $loginPath = '/login';

c) Click here to reset your password: {{ url('password/reset/'.$token) }}

d) 权限判断

```
<ul class="nav navbar-nav navbar-right">
    @if (Auth::guest())
        <li><a href="/login">登录</a></li>

        <li><a href="/auth/register">注册</a></li>
    @else
        <li class="dropdown">
            <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-expanded="false">{{ Auth::user()->name }} <span class="caret"></span></a>
            <ul class="dropdown-menu" role="menu">
                <li><a href="/auth/logout">注销登录</a></li>
            </ul>
        </li>
    @endif
</ul>
```

三十五、使用 Laravel 认证完成页面访问控制

a) 使用验证中间件完成访问页面控制

b) 路由配置

```
Route::group(['middleware' => 'auth'], function () {
    Route::resource('/yuyue','YuyueController',['only' => ['create','store']]);
});
```

c) 单个控制器路由配置

```
public function __construct()
{
      $this->middleware('auth', ['except' => 'index']);
}
```