

```
export GRADLE_HOME=/Applications/Android\ Studio.app/Contents/gradle/
gradle-3.2
export JAVA_HOME=/Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/
Contents/Home
export ANDROID_HOME=/Users/apple/Library/Android/sdk
export NODE_PATH=/usr/local/lib/node_modules
export HADOOP_HOME=/Users/apple/Hadoop/hadoop-2.8.1
export PATH=$PATH:~/composer/vendor/bin:/Applications/MAMP/bin/php/
php7.1.6/bin:$JAVA_HOME/bin:$HADOOP_HOME/bin:$HADOOP_HOME/sbin:
$GRADLE_HOME/bin:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools
```

Mac下升级PHP版本至7.1

博主在搞Web开发主要采用的是Laravel，然而发现其对PHP版本的要求是越来越高，PHP5.6已经越来越受到限制， Laravel 5.5将正式弃用PHP5.6，所以博主决定直接升级到7.1版本。

移除旧版本

由于系统本身已经装了PHP5.6，所以需要先将其移除。
在这里列出目录以及移除需要的命令。

/private/etc/	sudo rm -rf php-fpm.conf.default php.ini php.ini.default
/usr/bin/	sudo rm -rf php php-config phpdoc phpize
/usr/include	sudo rm -rf php
/usr/lib	sudo rm -rf php
/usr/sbin	sudo rm -rf php-fpm
/usr/share	sudo rm -rf php
/usr/share/man/man1	sudo rm -rf php-config.1 php.1 phpize.1
/usr/share/man/man8	sudo rm -rf php-fpm.8

顺次手动删除它们即可。

搞清关系

在卸载过程中你会发现有PHP、FastCGI、php-fpm、spawn-fcgi等等的概念，所以在这里先梳理一下。

CGI

CGI是为了保证web server传递过来的数据是标准格式的，方便CGI程序的编写者。web server（比如说nginx）只是内容的分发者。比如，如果请求/index.html，那么web server会去文件系统中找到这个文件，发送给浏览器，这里分发的是静态数据。好了，如果现在请求的是/index.php，根据配置文件，nginx知道这个不是静态文件，需要去找PHP解析器来处理，那么他会把这个请求简单处理后交给PHP解析器。Nginx会传哪些数据给PHP解析器呢？url要有吧，查询字符串也得有吧，POST数据也要有，HTTP header不能少吧，好的，CGI就是规定要传哪些数据、以什么样的格式传递给后方处理这个请求的协议。仔细想想，你在PHP代码中使用的用户从哪里来的。

当web server收到/index.php这个请求后，会启动对应的CGI程序，这里就是PHP的解析器。接下来PHP解析器会解析php.ini文件，初始化执行环境，然后处理请求，再以规定CGI规定的格式返回处理后的结果，退出进程。web server再把结果返回给浏览器。

FastCGI

Fastcgi是用来提高CGI程序性能的。

那么CGI程序的性能问题在哪呢？”PHP解析器会解析php.ini文件，初始化执行环境”，就是这里了。标准的CGI对每个请求都会执行这些步骤（不闲累啊！启动进程很累的说！），所以处理每个时间的时间会比较长。这明显不合理嘛！那么Fastcgi是怎么做的呢？首先，Fastcgi会先启一个master，解析配置文件，初始化执行环境，然后再启动多个worker。当请求过来时，master会传递给一个worker，然后立即可以接受下一个请求。这样就避免了重复的劳动，效率自然是高。而且当worker不够用时，master可以根据配置预先启动几个worker等着；当然空闲worker太多时，也会停掉一些，这样就提高了性能，也节约了资源。这就是fastcgi的对进程的管理。

PHP-FPM

是一个实现了Fastcgi的程序，被PHP官方收了。

大家都知道，PHP的解释器是php-cgi。php-cgi只是个CGI程序，他自己本身只能解析请求，返回结果，不会进程管理（皇上，臣妾真的做不到啊！）所以就出现了一些能够调度php-cgi进程的程序，比如说由lighttpd分离出来的spawn-fcgi。好了PHP-FPM也是这么个东东，在长时间的发展后，逐渐得到了大家的认可（要知道，前几年大家可是抱怨PHP-FPM稳定性太差的），也越来越流行。

php-fpm的管理对象是php-cgi。但不能说php-fpm是fastcgi进程的管理器，因为前面说了fastcgi是个协议，似乎没有这么个进程存在，就算存在php-fpm也管理不了他（至少目前是）。有的说，php-fpm是php内核的一个补丁

以前是对的。因为最开始的时候php-fpm没有包含在PHP内核里面，要使用这个功能，需要找到与源码版本相同的php-fpm对内核打补丁，然后再编译。后来PHP内核集成了PHP-FPM之后就方便多了，使用--enable-fpm这个编译参数即可。

安装PHP7.1

用brew进行安装。

```
brew install homebrew/php/php71  
brew install homebrew/php/php71-mcrypt
```

安装完了之后它会自带PHP-FPM，在
启动PHP-FPM

```
sudo php-fpm  
1 sudo php-fpm
```

配置文件目录

php.ini

/usr/local/etc/php/7.1/php.ini

php-fpm.conf

/usr/local/etc/php/7.1/php-fpm.conf

php-fpm

/usr/local/opt/php71/sbin/php-fpm

但是执行php-fpm发现没有反应，所以这里需要加一个symlink

```
ln -s /usr/local/opt/php71/sbin/php-fpm /usr/local/bin/php-fpm
```

然后运行php-fpm

```
sudo php-fpm
```

启动nginx

```
sudo nginx
```

关于MySQL和其他的安装在这就不再赘述。

以上便完成了PHP的升级。

AppleHDA Patche

```
update kti_collect set addr_detail_h=
concat(addr_sheng_h ,addr_shi_h,addr_xian_h,addr_detail_h)
```

1 安装 composer

打开命令行并依次执行下列命令安装最新版本的 Composer:

```
php -r "copy('https://install.phpcomposer.com/installer', 'composer-setup.php');"
```

```
php composer-setup.php
```

```
php -r "unlink('composer-setup.php');
```

上述 3 条命令的作用依次是：

1. 下载安装脚本 – composer-setup.php – 到当前目录。
2. 执行安装过程。

3. 删除安装脚本。

全局安装

全局安装是将 Composer 安装到系统环境变量 PATH 所包含的路径下面，然后就能够命令行窗口中直接执行 composer 命令了。

Mac 或 Linux 系统：

打开命令行窗口并执行如下命令将前面下载的 composer.phar 文件移动到 /usr/local/bin/ 目录下面：

复制

```
sudo mv composer.phar /usr/bin/composer
```

最后

提示：不要忘了经常执行 composer selfupdate 以保持 Composer 一直是最新版本哦！

1>全局安装laravel：

```
composer global require "laravel/installer"
```

我们安装的内容在~/.composer目录下

我们现在还不能使用laravel，我们还需要在\$PATH下配置一下

```
vi ~/.bash_profile
```

添加

```
export PATH=$PATH:~/composer/vendor/bin
```

使用命令刷新

```
source ~/.bash_profile
```

2>使用composer安装laravel

```
composer create-project laravel/laravel 项目名 --prefer-dist
```

-----启动Laravel服务-----

定位到项目文件夹下

1>

```
php -S localhost:8001 -t public/
```

这样访问localhost:8001就可以访问了

2>

```
php artisan serve
```

这样访问localhost:8000就可以访问了

直接用 Composer 创建 Laravel 项目

参照网上的方案，先执行加速 composer 的执行（用国内的镜像，好人呐！）：

执行以下3条命令就可以安装了

```
1 composer config -g repo.packagist composer https://packagist.phpcomposer.com
```

```
2 composer global require "laravel/installer"
```

```
3 larval new blog
```

```
如果不行可以执行
```

```
composer create-project --prefer-dist laravel/laravel blog
```

```
创建项目
```

测试一下

在浏览器里面查看，可以看到大大的 Laravel，OK了。

```
alias php="/Applications/mamp/php/bin/php"
```

安装 node-v6.11.1

安装完成后进入项目目录后运行 npm install 安装项目依赖。可能出现下面的问题：

```
#sudo npm i -g npm #更新npm
```

```
# npm install
```

```
npm WARN prefer global marked@0.3.6 should be installed with -g
```

```
npm WARN prefer global node-gyp@3.6.2 should be installed with -g
```

重新用下面的命令来安装就可以了

```
#npm install marked@0.3.6 -g
```

```
#npm install node-gyp@3.6.2 -g
```

```
#npm install vue --save
```

```
#npm install --save vue-router
```

```
#npm install stylus --save
```

```
#npm install stylus-loader --save
```

```
#npm install extract-text-webpack-plugin --save
```

```
#npm install fs --save
```

```
#npm install css-what --save
```

```
npm install normalize.js --save
```

```
npm install async.js --save
```

```
#npm install element-ui --save
```

```
##npm install cross-env --save-dev
```

```
# npm install 最后运行一次来检查是否出错
```

```
npm WARN deprecated CSSselect@0.7.0: the module is now available as 'css-select'
```

```
npm WARN deprecated CSSwhat@0.4.7: the module is now available as 'csswhat'
```

```
appledeiMac:mmblog apple$ npm install
```

```
npm WARN prefer global marked@0.3.6 should be installed with -g
```

npm WARN prefer global node-gyp@3.6.2 should be installed with -g

下面是vue的一些配置情况，根据实际项目使用

```
{  
  "private": true,  
  "scripts": {  
    "dev": "npm run development",  
    "development": "cross-env NODE_ENV=development node_modules/  
webpack/bin/webpack.js --progress --hide-modules --config=node_modules/  
laravel-mix/setup/webpack.config.js",  
    "watch": "cross-env NODE_ENV=development node_modules/webpack/bin/  
webpack.js --watch --progress --hide-modules --config=node_modules/  
laravel-mix/setup/webpack.config.js",  
    "watch-poll": "npm run watch -- --watch-poll",  
    "hot": "cross-env NODE_ENV=development node_modules/webpack-dev-  
server/bin/webpack-dev-server.js --inline --hot --config=node_modules/  
laravel-mix/setup/webpack.config.js",  
    "prod": "npm run production",  
    "production": "cross-env NODE_ENV=production node_modules/webpack/  
bin/webpack.js --progress --hide-modules --config=node_modules/laravel-  
mix/setup/webpack.config.js"  
  },  
  "devDependencies": {  
    "axios": "^0.16.2",  
    "bootstrap-sass": "^3.3.7",  
    "cross-env": "^5.0.1",  
    "jquery": "^3.1.1",  
    "laravel-mix": "^1.0",  
    "lodash": "^4.17.4",  
    "stylus-loader": "^3.0.1",  
    "vue": "^2.3.3",  
    "vue-router": "^2.3.1"  
  },  
  "dependencies": {  
    "extract-text-webpack-plugin": "^2.1.2",  
    "stylus": "^0.54.5"  
  }  
}
```

webpack.mix.js文件打包设置

```
//mix.js('resources/assets/js/app.js', 'public/js')  
// .sass('resources/assets/sass/app.scss', 'public/css');  
mix.js('resources/assets/js/hello.js', 'public/js')  
  .extract(['lodash','jquery','axios','vue','vue-router']);
```

Npm run dev 运行打包编译

Php artisan serve 打开网站服务进行调试

MAC下安装MAMP的PHPredis扩展

1. 下载phpredis扩展安装包。git clone <https://github.com/nicolasff/phpredis.git>;
2. 解压后，进入该目录cd phpredis;
3. 依次执行以下操作完成安装；
4. /Applications/MAMP/bin/php/php5.6.30/bin/phpize
5. ./configure --with-php-config=/Applications/MAMP/bin/php/php5.6.30/bin/php-config
6. make
7. make install
8. 安装成功后redis.so会复制到/Applications/MAMP/bin/php/php5.6.30/lib/php/extensions/no-debug-non-zts-20131226目录下。如果不存在，手动将phpredis/modules目录下的redis.so复制过去。至此，phpredis扩展已安装成功。
9. 配置mamp php.ini；

```
/Applications/mamp/bin/php/php5.6.30/bin/phpize  
.configure --with-php-config=/Applications/MAMP/bin/php/php5.6.30/bin/php-config  
Make  
Make install 安装，如果安装不成功 可以用下面的手动复制过去  
请注意，上面的目录路径有可能跟我的不一样，请自行核对并调整，我在这里再次入坑  
编译成功后会在phpredis/modules下生成redis.so，把它复制到applications/  
MAMP/bin/php/php5.6.10/lib/php/extensions/no-debug-non-zts-****这个目录  
6、修改 php.ini  
打开 mamp->file->edit template->php 5.6.30 php.ini  
搜索 "extension="  
在后面添加一行"extension=redis.so"，保存后重启 mamp
```

7、检查组件是否安装成功

新建一个 **php** 文件，输入

```
<?php phpinfo(); ?>
```

保存后运行，查看是否有 **redis** 关键字

1、安装 redis 服务

下载并解压 <http://download.redis.io/releases/redis-3.2.6.tar.gz>

在终端下切换到你对应的目录

```
cd redis-3.2.6
```

```
make
```

当然你也可以使用 **brew** 安装，更方便快捷

2、启动服务

src/redis-server

安装redis服务端。我是用的brew安装的，蛮方便。

1. brew install redis
2. redis-server /usr/local/etc/redis.conf
3. redis-cli ping

8、测试 redis

新建 php 文件，输入

```
<?php
$redis = new redis();
$redis->connect('127.0.0.1', 6379);
$redis->set('Magic','http://shejishi.cc');
echo $redis->get('Magic');
?>
```

五、启动redis客户端实测

新开个终端，进入到src文件夹，执行命令 如图

```
./redis-cli //表示启动redis客户端
set admin echoRedis // 设置 key为admin value为echoRedis (redis的set语法)
//返回结果OK为 设置成功
get admin // 取 key为admin的value值 (redis的get语法)
//返回结果为 echoRedis (我们设置的值)
【表示redis客户端测试成功】
```

如果/Applications/MAMP/bin/php/php5.6.30/bin/phpize 出错就要安装autoconf
安装下载autoconf-2.69.tar安装 autoconf 并安装：

```
cd autoconf-2.69
./configure && make
sudo make install
```

创建数据库迁移

- 1 php artisan make:migration create_posts_table
- 2 到database/migrations目录下相看生成的代码文件 进行更改 后 执行下面的命令去建表
- 3 创建表 php artisan migrate

数据填充到database/factories/modelFactory.php目录下 参照里面的示例内容进行

```
$factory->define(App\Post::class, function (Faker\Generator $faker) {
    return [
        'title' => $faker->sentence(6),//6个单词
        'content' => $faker->paragraph(10),//10个句子
    ];
});
```

```
];
});

Php artisan tinker //启动tinker
factory(App\Post::class,10)->make(); //在屏幕上打印出来
factory(App\Post::class,10)->create(); //写入数据库
```

创建模型

```
php artisan make:model Post
```

打开tinker 手动往表里增加数据，主要用于测试自己生的模型中的类使用对不对
php artisan tinker

```
$post = new \App\Post(); //new一个post
```

示例如下：

```
appledeiMac:mmblog apple$ php artisan tinker
Psy Shell v0.8.9 (PHP 5.6.30 — cli) by Justin Hileman
>>> $post = new \App\Post();
=> App\Post {#695}
>>> $post->title = "this is post1";
=> "this is post1"
>>> $post->content = "this is post1 content";
=> "this is post1 content"
>>> $post->save();
=> true
```

```
>>> \App\Post::find(2); //按主键id查找命令 返回的是数组
=> App\Post {#704
  id: 2,
  title: "this is post2",
  content: "this is post2 content2",
  user_id: 0,
  created_at: "2017-07-27 08:41:06",
  updated_at: "2017-07-27 08:41:06",
}
```

上面的查找出来后 可以直接用下面的指令进行修改 同增加

```
$post->title = "this is post1";
```

```
$post->save();
```

```
>>> $post = \App\Post::find(2);
=> App\Post {#689
  id: 2,
  title: "this is post2",
  content: "this is post2 content2",
  user_id: 0,
  created_at: "2017-07-27 08:41:06",
  updated_at: "2017-07-27 08:41:06",
}
```

```
>>> $post->delete(); //直接删除
=> true

>>> \App\Post::where('title','this is post1')->first(); //字条件查找 返回的是数组
=> \App\Post {#708
    id: 1,
    title: "this is post1",
    content: "this is post1 content",
    user_id: 0,
    created_at: "2017-07-27 08:36:14",
    updated_at: "2017-07-27 08:36:14",
}

>>> \App\Post::where('title','this is post1')->get(); //返回的是对象
=> \Illuminate\Database\Eloquent\Collection {#705
    all: [
        \App\Post {#706
            id: 1,
            title: "this is post1",
            content: "this is post1 content",
            user_id: 0,
            created_at: "2017-07-27 08:36:14",
            updated_at: "2017-07-27 08:36:14",
        },
    ],
}
```

更改laravel5.4的时区 config/app.php
'timezone' => 'UTC', 改成中国时区就可以了. **'timezone' => 'PRC'**,

//创建文件上传位置的链接地址 (config/filesystems.php)

修改

```
//'default' => env('FILESYSTEM_DRIVER', 'local'),
'default' => env('FILESYSTEM_DRIVER', 'public'),
```

然后执行

Php artisan storage:link

这样文件上传图版的位置就是/storage/app/public

从 \$app=app () 获取容器

```
$log=$app->make("log"); //绑定对象
$log->info(); //使用对象
```

```
$app = app();
$log = $app->make('log');
```

```
$log->info("post_index",['data'=>'this is post index']);
```

```
# tail -f storage/logs/laravel.log. 查看日志
```

用tinker 查看对象

```
appledeiMac:mmblog apple$ php artisan tinker
Psy Shell v0.8.9 (PHP 5.6.30 — cli) by Justin Hileman
>>> app('Psr\Log\LoggerInterface');
=> Illuminate\Log\Writer {#696}
>>> app('log');
=> Illuminate\Log\Writer {#696}
>>>
```

把对象查出来了，可以直接 查看对象文档的接口，就知道如何使用对象了

Command +SHIFT+O Phpstorm查看指定的文件

前台不准备使用AUTH组件，因为前台登陆注册 复杂一些 所以我们删除auth 目录中的auth组件控制器

```
php artisan make:controller RegisterController
php artisan make:controller LoginController
php artisan make:controller UserController
```

重新建立user模型， 使用

```
use Illuminate\Foundation\Auth\User as Authenticatable;
class User extends Authenticatable
```

1建立对文章的权限策略(生成的类在app/Policies目录下)

```
php artisan make:policy PostPolicy
```

```
<?php
```

```
namespace App\Policies;
```

```
use App\User;
use App\Post;
use Illuminate\Auth\Access\HandlesAuthorization;
```

```
class PostPolicy
{
    use HandlesAuthorization;
```

```
/*
 * Create a new policy instance.
```

```

*
* @return void
*/
public function __construct()
{
    //
}
//修改 哪些人可以 修改 文章
public function update(User $user,Post $post)
{
    return $user->id == $post->user_id;
}

//删除 哪些人可以 删除 文章
public function delete(User $user,Post $post)
{
    return $user->id == $post->user_id;
}
}

```

2注册文章策略类(App/Providers/AuthServiceProvider.php)

<?php

namespace App\Providers;

```

use Illuminate\Support\Facades\Gate;
use Illuminate\Foundation\Support\Providers\AuthServiceProvider as
ServiceProvider;

```

class AuthServiceProvider **extends** ServiceProvider

{

```

/**
 * The policy mappings for the application.
 *
 * @var array
 */

```

protected \$policies = [

```

    // 'App\Model' => 'App\Policies\ModelPolicy',
    'App\Post' => 'App\Policies\PostPolicy',
];
```

/**

* Register any authentication / authorization services.

*

* @return void

*/

```
public function boot()
{
    $this->registerPolicies();

    //
}
```

3 使用策略类 在文章控制器update操作更新前 使用如下语句

```
$this->authorize('update',$post);//使用权限策略
```

//2逻辑

```
$post->title = request('title');
$post->content = request('content');
$post->save();
```

4 在模板中 也要使用 检查 让没有权限的用户的功能按键不能显示

```
@can('update',$post)
<a style="margin: auto" href="/posts/{{ $post->id}}/edit">
    <span class="glyphicon glyphicon-pencil" aria-hidden="true"></span>
</a>
@endcan
@can('delete',$post)
<a style="margin: auto" href="/posts/{{ $post->id}}/delete">
    <span class="glyphicon glyphicon-remove" aria-hidden="true"></span>
</a>
@endcan
```

php artisan make:migration create_comments_table. //建立数据表迁移

```
<?php
```

```
use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
```

```
class CreateCommentsTable extends Migration
{
```

```
    /**
     * Run the migrations.
     *
```

```
     * @return void
     */
```

```
public function up()
```

```
{
```

```
    Schema::create('comments', function (Blueprint $table) {
        $table->increments('id');
```

```



```

php artisan migrate/建立数据表

解压elasticsearch包后，进入解压后的目录

使用elasticsearch 中文查询

主要用到analysis-ik

#bin/elasticsearch-plugin list 显示所有的组件

#bin/elasticsearch-plugin list >/tmp/plugin.log 显示所有的组件到/tmp/plugin.log

把这个文件中的要的analysis-ik删掉后，运行下面的语句，就去除了 不常用的插件

#cat /tmp/plugin.log|xargs -I {} bin/elasticsearch-plugin remove {}

bin/elasticsearch -d 来起用

在浏览器中输入下面地址看看有没有反映

<http://127.0.0.1:9200/> 或者 vim logs/elasticsearch.log 查看启动日志，可以看到启动状态

A安装laravel 使用elastic的包

1 进入项目目录

 composer require laravel/scout

2. App/config/app.php 增加下面一行

 Laravel\Scout\ScoutServiceProvider::class,

3. 执行以下安装

 php artisan vendor:publish --
 provider="Laravel\Scout\ScoutServiceProvider"

B安装scout的es驱动

 composer require tamayo/laravel-scout-elastic

App/config/app.php 增加下面一行

 ScoutEngines\Elasticsearch\ElasticsearchProvider::class,

C修改scout.php

修改 'driver' => env('SCOUT_DRIVER', 'elasticsearch'),
并在最后增加驱动如下：
'elasticsearch' => [
 'index' => env('ELASTICSEARCH_INDEX', 'laravel54'),
 'hosts' => env('ELASTICSEARCH_HOST', 'http://127.0.0.1:9200'),
,

创建laravel自定义命令行

composer require guzzlehttp/guzzle 引入
php artisan make:command ESInit 会生成在 app/commands/ESinit.php文件
并修改如下内容
在前面引入use GuzzleHttp\Client;

```
protected $signature = 'es:init';
protected $description = 'init laravel es for post';
```

handle函数如下

```
public function handle()
{
    //创建template
    $client = new Client();
    $url = config('scout.elasticsearch.hosts')[0].'/_template/tmp';
    $client->delete($url);
    $param = [
        'template' => config('scout.elasticsearch.index'),
        'mappings' => [
            '_default_' => [
                'dynamic_templates' => [
                    'strings' => [
                        'type' => 'text',
                        'analyzer' => 'ik_smart',
                        'fields' => [
                            'keyword' => [
                                'type' => 'keyword'
                            ]
                        ]
                    ]
                ]
            ],
        ];
    ];
    $client->put($url, $param);
    $this->info("=====创建模板成功");
}
```

```

=====
//创建索引
$url = config('scout.elasticsearch.hosts')
[0]:'/.config('scout.elasticsearch.index');
$client->delete($url);
$param=[

'json' => [
    'settings' => [
        'refresh_interval' => '5s',
        'number_of_shards' => 1,
        'number_of_replicas' => 0,
    ],
    'mappings' => [
        '_default_' => [
            '_all' => [
                'enabled' => false
            ]
        ]
    ]
]
];
$client->put($url,$param);
$this->info("=====创建索引成功
=====");
}

```

打开app/console/kernel.php

```

protected $commands = [
    //在此加入挂载的命令类就可以了
    \App\Console\Commands\ESInit::class
];

```

现在用php artisan命令 可以看到有如下的命令信息行显示

```

es
  es:init      init laravel es for post
说明我们的命令配置成功了

```

运行命令看看是不是可以执行成功 php artisan es:init

导入POST模型数据， 使用以下命令 后面跟的是具体的模型

```
php artisan scout:import "\App\Post"
```

提示如下 说明导入成功

```
Imported [\App\Post] models up to ID: 35
```

All [\App\Post] records have been imported.

可以使用如下命令来查看是否有查询结果

后面的参数是索引/类型/编号

<http://127.0.0.1:9200/laravel54/post/35>

模型的scope

后台第一个用户建立

```
appledeiMac:mmblog apple$ php artisan tinker
Psy Shell v0.8.9 (PHP 5.6.30 — cli) by Justin Hileman
>>> $admin = new \App\AdminUser;
=> \App\AdminUser {#734}
>>> $admin->name="test1";
=> "test1"
>>> $admin->password=bcrypt("123456");
=> "$2y$10$HdGemixPPNOYB1LF3HvlweswLFatV8NQt/
pzHVnX251EjEEvNrQ06"
>>> $admin->save();
=> true
>>>
```

php artisan make:migration alter_posts_table //修改表结构

```
public function up()
{
    Schema::table("posts",function (Blueprint $table){
        $table->tinyInteger('status')->default(0); //文章状态 0未知 1通过 -1删除
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table("posts",function (Blueprint $table){
        $table->dropColumn('status');
    });
}
php artisan migrate //更新表的迁移
```

```
Collection {#263 ▼
  #items: array:2 [▼
    0 => AdminPermission {#262 ►}
    1 => AdminPermission {#265 ►}
  ]
}
```

```
BelongsToMany {#256 ▼
  #table: "admin_permission_role"
  #foreignKey: "role_id"
  #relatedKey: "permission_id"
  #relationName: "permissions"
  #pivotColumns: array:2 [►]
  #pivotWheres: []
  #pivotWhereIns: []
  #pivotCreatedAt: null
  #pivotUpdatedAt: null
  #using: null
  #query: Builder {#261 ►}
  #parent: AdminRole {#252 ►}
  #related: AdminPermission {#255 ►}
}
```

```
$('label[class="checkbox"]').on('mouseleave',function(){//æ®æ£é€
%oæ<@ä, æ•°æ~¾ç¤º var allv1 = $('.multiselect-container >.active > a
>.checkbox >input'); var len1 =allv1.length; $('.multiselect-selected-
text').text(len1); });

$('.dropdown-toggle').on('click',function(){//æ‰æ•°æ®åŠ”å¬¹åŽ¥ä»žæ•°æ®åº“ä,-
æ¥çš„æ•°æ®è¿›è;Œé™,,å€¼ var allv = \$\('option\[isactive="1"\]'\); //var len
=allv.length; // \$\('.multiselect-selected-text'\).text\(len+'ä, æ€‰æ<@'\);
allv.each\(function\(index\) { //console.log\(\$\(this\).attr\('value'\)\); var n = \$
\(this\).attr\('value'\); \$\('input\[value='+n+'\]'\).closest\('li'\).addClass\('active'\); \$
\('input\[value='+n+'\]'\).prop\("checked",true\); \$
\('input\[value='+n+'\]'\).closest\('label'\).prop\('selected',true\); if\(index>1\){ \$
\('button\[title\]'\).attr\('title', \$\('button\[title\]'\).attr\('title'\)+','+n\); }else{ \$
\('button\[title\]'\).attr\('title', n\); } \$\('.multiselect-selected-text'\).text\(index\); }\);
return false; }\);

\$ v \) { ?> >
```

```
<?php
Use ZPHP\ZPHP;
```

```

Use ZPHP\Core\Config as ZConfig;
Use ZPHP\Core\Factory as Factory;
Class HttpServer
{
    Private static $instance;
    Public static $server;
    Public static $request;
    Public static $response;
    Public static $wsfarme;
    Public static $http;
    Private $zphp;
    Private $webPath;
    Private $defaultFiles = ['index.html', 'main.html', 'default.html'];
    Private $configPath='default';
    Private $mimes = [];

    Public function __construct($opt,$config ='default')
    {
        $this->webPath = $opt['path'];
        if(!empty($opt['config'])) {
            $this->configPath = $opt['config'];
        }
        $ip = empty($opt['ip'])? '0.0.0.0' : $opt['ip'];
        $port = empty($opt['port'])? '9501' : $opt['port'];
        $http = swoole_http_server($ip,$port);
        Self::$wsfarme = new swoole_websocket_frame();
        if(isset($opt['d'])) {
            $damemonize =1;
        }else{
            $damemonize =0;
        }
        $worker_num = empty($opt['worker']) ? 4 : $opt['worker'];
        $http->set(
            array(
                'worker_num' => $worker_num,
                'demonize' => $demonize,
                'max_request' => 0,
            )
        );
        $http-
>setGlobal(HTTP_GLOBAL_ALL,HTTP_GLOBAL_GET,HTTP_GLOBAL_POST);
        $http->on('WorkerStart',array($this,'onWorkStart));
        $http->on('WorkerError',array($this,'onWorkError));
        $http->on('WorkerStop',array($this,'onWorkStop));
    }
}

```

```

$http->on('close',function() {
    $params = func_get_args();
    $conn = $params[0] ->connection_info($params[1]);
    if($conn['websocket_status'] >1) {
        $parse =
ZFactory::getInstance(Config::getField('socket','parse_class','WebSocketChat
Parse'));
        $parse->close($this->php,$params[1]);
    }
});

$http->on('open',function($response) {
    $parse =
ZFactory::getInstance(Config::getField('socket','parse_class','WebSocketChat
Parse'));
    $conn = $params[0] ->connection_info($params[1]);
    $parse->open($this->php,$response->fd);

});

$http->on('message',function($frame) {
    $parse =
ZFactory::getInstance(Config::getField('socket','parse_class','WebSocketChat
Parse'));
    $conn = $params[0] ->connection_info($params[1]);
    $parse->message($this->php,$frame);

});

$http->on('request',function($request,$response) {
    HttpServer::$request = $request;
    HttpServer::$response = $response;
    $_SERVER['PATH_INFO'] = $request->server['path_info'];
    if( $_SERVER['PATH_INFO'] == '/'){
        if(!empty($this->defaultFiles)){
            Foreach ($this->defaultFiles as $file){
                $staticFile = $this->getStaticFile(DIRECTORY_SEPARATOR,$file);
                if(is_file($staticFile)) {
                    $response->end(file_get_contents($staticFile));
                    return;
                }
            }
        }
        if( $_SERVER['PATH_INFO'] == '/'){
            $response->header('Content-Type',$this->mines['ico']);
            $response->end('');
            return;
        }
    }
})

```

```

$staticFile = $this->getStaticFile( $_SERVER['PATH_INFO'] );
if(\is_dir( $staticFile)){
    Foreach ($this->defaultFiles as $file){
        if(is_file($staticFile.$file)) {
            $response->header('Content-Type','text/html');
            $response->end(file_get_contents($staticFile.$file));
            return;
        }
    }
}
$ext = \pathinfo($_SERVER['PATH_INFO'],PATHINFO_EXTENSION);
if(isset($this->mimes[$ext])){
    if(is_file($staticFile)) {
        $response->header('Content-Type',$this->mimes[$ext]);
        $response->end(file_get_contents($staticFile));
        return;
    }else{
        $response->status(404);
        $response->end("");
        return;
    }
}
try {
    ob_start();
    $result = $this->php->run();
    if(null == $result ){ $result = ob_get_contents();}
    ob_end_clean();
}catch (Exception $exception){
    $result =
call_user_func(Config::getField('project','exception_handler','ZPHP\ZPHP::exceptionHandler'),$exception);
}
$response->status(200);
$response->end($result);

});

Self::$http = $http;
Self::$http->start();

```

```

}

Public function onWorkerStart()
{
    opcache_reset();
    Require
dirname(__DIR__).DIRECTORY_SEPARATOR .'ZPHP'.DIRECTORY_SEPARATOR .'p
hp.php';
    $this->php =ZPHP::run($this->webPath,false,$this->configPath);
    $params = func_get_args();
    $this->mimes = require 'mimes.php';
}

Public function onWorkerStop()
{
    $params =(func_get_args());
}

Public function onWorkerStart()
{
    $params =(func_get_args());
}

Public static getInstance($webPath,$config='default')
{
    if(!self::$instance){
        Self::$instance = new HttpServer($webPath,$config);

    }
    Return self::instance;

}

Private function getStaticFile($file,$path='webroot')
{
    Return $this->webPath . DIRECTORY_SEPARATOR .$path . $file;
}

define('USE_SWOOLE_HTTP_SERVER',1);
$opt = getopt("d",
    [
        'path:',
        'ip:',
        'port:', 'worker:', 'config:',
    ]);
if(empty($opt['path'])){
    Echo "examples : php swoole_http_server.php —path=/home/www/zphpdemo
}

```

```
—config=default —ip=0.0.0.0 —work=4 “;
Echo “path is required”.PHPEOL;
Return;

}
```

```
HttpServer::getInstance($opt);
```

```
composer global require 'phpunit/phpunit'
```

```
<?php
namespace Laravoole\Test;

use Adoy\FastCGI\Client as FastCgiClient;
use WebSocket\Client as WebSocketClient;

class LaravooleTest extends \PHPUnit_Framework_TestCase
{
    protected static $codeCoveraging = false;

    protected $handlers = [
        'SwooleHttp' => [
            'port' => 9050,
            'deal_with_public' => true,
            'gzip' => true,
            'daemonize' => false,
            'worker_num' => 1,
        ],
        // 'SwooleFastCGI' => [
        //     'port' => 9051,
        //     'daemonize' => false,
        //     'worker_num' => 1,
        //     'max_request' => 2000,
        // ],
        'SwooleWebSocket' => [
            'port' => 9052,
            'daemonize' => false,
            'worker_num' => 1,
        ],
        // 'WorkermanFastCGI' => [
    ];
```

```

        //  'port' => 9053,
        //  'daemonize' => true,
        //],
    ];
};

protected $fastCgiParams = [
    'GATEWAY_INTERFACE' => 'FastCGI/1.0',
    'REQUEST_METHOD' => 'GET',
    'SCRIPT_FILENAME' => '/index.php',
    'SERVER_SOFTWARE' => 'php/fcgiclient',
    'REMOTE_ADDR' => '127.0.0.1',
    'REMOTE_PORT' => '9985',
    'SERVER_ADDR' => '127.0.0.1',
    'SERVER_PORT' => '80',
    'SERVER_NAME' => 'mag-tured',
    'SERVER_PROTOCOL' => 'HTTP/1.1',
    'CONTENT_TYPE' => 'application/x-www-form-urlencoded',
    'CONTENT_LENGTH' => 0,
];
;

public function testSwooleHttpCreate()
{
    $this->createServer('SwooleHttp');
}

/*
 * @depends testSwooleHttpCreate
 */
public function testSwooleHttp()
{
    $this->assertRequests('SwooleHttp', 'http', function ($uri) {
        return $this->requestHttp('http://localhost:9050' . $uri);
    });
}

/*
 * @depends testSwooleHttpCreate
 */
public function testSwooleHttpClose()
{
    $this->closeSwoole('SwooleHttp');
}

//public function testSwooleFastCgiCreate()
//{
//    $this->createServer('SwooleFastCGI');

```

```

//}

/**
 * @depends testSwooleFastCgiCreate

public function testSwooleFastCgi($client)
{
    // $this->assertRegExp('~Laravel~', $this->requestFastCgi('http://
localhost:9051/'));

    $this->assertRequests('SwooleFastCGI', 'raw', function ($uri) {
        return $this->requestFastCgi('http://localhost:9051' . $uri);
    });
} */

/**
 * @depends testSwooleFastCgiCreate

public function testSwooleFastCgiClose()
{
    $this->closeSwoole('SwooleFastCGI');
} */

public function testSwooleWebSocketCreate()
{
    $this->createServer('SwooleWebSocket');
}

/**
 * @depends testSwooleWebSocketCreate
 */
public function testSwooleWebSocket()
{
    $client = new WebSocketClient('ws://localhost:9052', [
        'headers' => [
            'Sec-WebSocket-Protocol' => 'jsonrpc',
        ],
    ]);

    for ($id = 1; $id < 3; $id++) {
        $this->assertJsonStringEqualsJsonString($this-
>requestWebSocket($client, '/json', $id), json_encode([
            'status' => 200,
            'method' => '/json',
            'result' => json_encode([
                'object' => ['property' => 'value'],
            ]),
        ]));
    }
}

```

```

        'array' => ['foo', 'bar'],
    ]),
    'id' => $id,
    'error' => null,
]);
}

$this->assertRequests('SwooleWebSocket', 'http', function ($uri) {
    return $this->requestHttp('http://localhost:9052' . $uri);
});

/** 
 * @depends testSwooleWebSocketCreate
 */
public function testSwooleWebSocketClose()
{
    $this->closeSwoole('SwooleWebSocket');
}

//public function testWorkermanFastCgiCreate()
//{
//    $this->createServer('WorkermanFastCGI');
//}

/** 
 * @depends testWorkermanFastCgiCreate
 */
public function testWorkermanFastCgi($client)
{
    // sleep(30);
    $this->assertRequests('WorkermanFastCGI', 'raw', function ($uri) {
        return $this->requestFastCgi('http://localhost:9053' . $uri);
    });
} */

/** 
 * @depends testWorkermanFastCgiCreate
 */
public function testWorkermanFastCgiClose()
{
    $this->closeWorkerman('WorkermanFastCGI');
} */

protected function closeSwoole($mode)
{

```

```

$pid = (int) @file_get_contents($pidFile = $this->getPidFilePath($mode));
if ($pid && posix_kill($pid, SIGTERM)) {
    @unlink($pidFile);
}
}

//protected function closeWorkerman($mode)
//{
//  $pid = (int) @file_get_contents($pidFile = $this-
>getPidFilePath($mode));
//  if ($pid && posix_kill($pid, SIGINT)) {
//      @unlink($pidFile);
//  }
//}

protected function assertRequests($mode, $resultType, $callback)
{
    $this->assertRegExp('~Laravel~', $callback('/'));
    $this->assertStringEndsWith('Laravoole', $callback('/laravoole'));
    $this->assertRegExp('~Laravel - A PHP Framework For Web Artisans~',
$resultType, $callback('/download'));
    if (isset($this->handlers[$mode]['deal_with_public'])) && $this-
>handlers[$mode]['deal_with_public']) {
        $this->assertStringStartsWith('User-agent', $callback('/robots.txt'));
    }

    if (self::$codeCoveraging) {
        $result = $callback('/codeCoverage');
        if ($resultType == 'raw') {
            $result = substr($result, 4 + strpos($result, "\r\n\r\n"));
        }
        $result = json_decode($result, true);
        $this->getTestResultObject()->getCodeCoverage()->append($result);
    }
}

protected function requestHttp($url)
{
    return file_get_contents($url);
}

/*protected function requestFastCgi($url)
{
    $components = parse_url($url);
    $client = new FastCgiClient($components['host'], $components['port']);
    $params = $this->fastCgiParams;
}

```

```

$params['REQUEST_URI'] = $components['path'];
$result = $client->request($params, '');
$client->setKeepAlive(false);
return $result;
}*/



protected function requestWebSocket($client, $uri, $id)
{
    $client->send(json_encode([
        'method' => $uri,
        'params' => [],
        'id' => $id,
    ]));

    return $client->receive();
}

protected function createServer($mode)
{
    $descriptorspec = array(
        0 => array("pipe", "r"), // stdin is a pipe that the child will read from
        // 1 => array("pipe", "w"), // stdout is a pipe that the child will write to
        // 2 => array("file", "/tmp/error-output.txt", "a") // stderr is a file to write
    to
    );
}

$config = $this->getConfig($mode);
$entry = 'Entry.php';

if (is_object($codeCoverage = $this->getTestResultObject()->getCodeCoverage()) {
    self::$codeCoveraging = true;
    $virus = function () {
        return $this->driver;
    };
    $virus = \Closure::bind($virus, $codeCoverage, $codeCoverage);
    $driver = $virus();
    if ($driver instanceof
        \SebastianBergmann\CodeCoverage\Driver\Xdebug || $driver instanceof
        \PHP_CodeCoverage_Driver_Xdebug) {
        $entry = 'XdebugEntry.php';
    } else {
        $entry = 'PHPDBGEntry.php';
    }
    $config['base_config']['code_coverage'] = get_class($driver);
}

```

```

$process = proc_open(PHP_BINARY . ' ' . __DIR__ . "/Entries/{$entry}
{$mode}", $descriptorspec, $pipes);

$this->assertTrue(is_resource($process));
$write = serialize($config);

fwrite($pipes[0], $write);
fclose($pipes[0]);
// fclose($pipes[1]);

// ensure server begin
sleep(1);
ini_set('default_socket_timeout', 1);
}

protected function getPidFilePath($mode)
{
    return __DIR__ . "/../{$mode}.pid";
}

protected function getConfig($mode)
{
    $wrapper = "Laravoole\\Wrapper\\{$mode}Wrapper";
    $wrapper_file = "src/Wrapper/{$mode}Wrapper.php";
    $handler_config = $this->handlers[$mode];
    $port = $handler_config['port'];
    unset($handler_config['port']);

    $laravooleConfig = include __DIR__ . '/../config/laravoole.php';
    $base_config = $laravooleConfig['base_config'];
    $base_config['callbacks']['bootstraped'][] = [Callbacks::class,
'bootstrapedCallback'];
    $base_config['callbacks']['bootstraping'][] = [Callbacks::class,
'bootstrapingCallback'];
    $wrapper_config = $laravooleConfig['wrapper_config'];
    $wrapper_config['environment_path'] = __DIR__ . '/..';

    $configs = [
        'host' => '127.0.0.1',
        'port' => $port,
        'wrapper_file' => $wrapper_file,
        'wrapper' => $wrapper,
        'pid_file' => $this->getPidFilePath($mode),
        'root_dir' => __DIR__ . '/../vendor/laravel/laravel',
        // for swoole / workerman
    ];
}

```

```

'handler_config' => $handler_config,
// for wrapper, like http / fastcgi / websocket
'wrapper_config' => $wrapper_config,
'base_config' => $base_config,
];

return $configs;
}

}

```

1. 我想说五个点

- 1. swoole_process 的创建默认是创建的管道，当想用消息队列时，记得把参数设成 **false**（其实我发现不写也行）
- 2. useQueue 要在 start 的之前调用，
- 3. pop 方法 默认是阻塞的
- 4. 一定一定要写 **wait**（这里坑死不少人）

微信小程序

控制显示

```

<view wx:if="{{show}}">{{text}}</view>
<view wx:else>2222222</view>
for 循环
<view wx:for="{{news}}" wx:for-item="newitem">
    {{index}}--{{newitem}}
</view>

```

//按钮事件邦定

```
<button type="primary" bindtap='btnClick'>{{btnText}}</button>
```

header.wxml

```
<text>这里是头部</text>
```

```
<include src="../templates/header" />
```

footer.wxml

```
<template name="footer1">
这是底部内容1{{text}}
</template>
<template name="footer2">
这是底部内容2{{text}}
</template>
<template name="footer3">
这是底部内容3{{text}}
</template>
```

```
<!--! 模板文件只能通过import导入 不能include !-->
<import src="../templates/footer" />
//指定使用那个模板
<template is="footer2" data="{{text:'导入设置的内容。。。。'}}" />
```

Bindtap 事件 触发冒泡事件
catchtap 事件 不触发冒泡事件

```
<view class="view1" bindtap="view1click">
  view 11
<view class="view2" bindtap="view2click">
  view 22
  <view class="view3" catchtap="view3click">
    view 33
  </view>
</view>
</view>
```

```
//事件处理函数
view1click:function () {
  console.log('view1click')
},
view2click: function () {
  console.log('view2click')
},
view3click: function () {
  console.log('view3click')
},
```

```
//绑定数据的方法 用data- dataset:{id: "100", title: "标题"}
<view class="view1" bindtap="view1click" id="view1" data-title="标题" data-
id="100">
```

```
view 11
<view class="view2" bindtap="view2click" id="view2">
    view 22
    <view class="view3" bindtap="view3click" id="view3">
        view 33
    </view>
</view>
</view>

{
    "pages": [
        "pages/index/index",
        "pages/logs/logs"
    ],
    "window": {
        "backgroundTextStyle": "light",
        "navigationBarBackgroundColor": "#fff",
        "navigationBarTitleText": "WeChat",
        "navigationBarTextStyle": "black"
    },
    "tabBar": { //必须与pages的第一项 对应， 不对应就不会显示
        "list": [
            {
                "pagePath": "pages/index/index",
                "text": "首页"
            },
            {
                "pagePath": "pages/logs/logs",
                "text": "日志"
            }
        ]
    }
}

"tabBar": {
    "selectedColor": "#11cd6e",
    "borderStyle": "white",
    "list": [
        {
            "iconPath": "images/icon1.png",
            "selectedIconPath": "images/icon1s.png",
            "pagePath": "pages/index/index",
            "text": "首页"
        }
    ]
}
```

```
},
{
  "iconPath": "images/icon2.png",
  "selectedIconPath": "images/icon2s.png",
  "pagePath": "pages/logs/logs",
  "text": "日志"
}
]
}
```

```
//页面跳转
itemClick: function () { //当前的页面还存在，可以返回
  wx.navigateTo({
    url: '../logs/logs',
  })
  wx.redirectTo({ //当前的页面就没有了
    url: '../logs/logs',
  })
}

itemClick: function () {
  wx.navigateTo({
    url: '../logs/logs?id=100'
  })
}

<navigator url="../logs/logs?id=100" redirect> //redirectTo
<view>
  <text class="user-motto">文章1</text>
</view>
</navigator>

<navigator url="../logs/logs?id=100" >/navigateTo
<view>
  <text class="user-motto">文章1</text>
</view>
</navigator>
```

wx.navigateTo,wx.redirectTo不能是跳到tabBar 中的页面

微信小程序UI精讲之布局基础

- flex布局基础
- 相对定位和绝对定位

主轴
交叉轴

flex容器属性详解

- flex-direction 决定元素的排列方向
- flex-wrap 决定元素如何换行（排列不下时）
- flex-flow flex-direction和flex-wrap的简写
- justify-content 元素在主轴上的对齐方式
- align-items 元素在交叉轴的对齐方式

flex元素属性详解

- flex-grow 当有多余空间时，元素的放大比例
- flex-shrink 当空间不足时，元素的缩小比例
- flex-basis 元素在主轴上占据的空间
- flex 是grow、shrink、basis的简写
- order 定义元素的排列顺序
- align-self 定义元素自身的对齐方式

comand+/ 注解

flex-direction 决定主轴与交叉轴 的方向

/*flex-direction: column;*/ / 主轴为纵坐标轴（从上到下）
/* flex-direction: row; */ 主轴为横坐标轴（从左到右） 这个是flex 布局的默认
方式
flex-wrap: wrap; 一行占不下时如何 ,wrap 换行
flex-wrap: nowrap; 不换行
flex-wrap: wrap-reverse; 反转 从最后开始

/*flex-direction: column;*/
/* flex-direction: row; */
/* flex-wrap: wrap-reverse; */
flex-flow: row nowrap //是 flex-direction 与flex-wrap:的简写

justify-content: center; //在主轴上的对齐方式 (flex-start 从左开始对齐 flex-end
右对齐 space-around在主轴上的空间是一样的 space-between 两端对齐
align-items: flex-start 在交叉轴的对齐方式 flex-start 左（上）对齐（默认方
式），flex-end右（下\尾部）对齐，center
align-items: stretch; //当元素没有高度时， 自动占满容器高度

align-items: baseline; //以元素文字对齐

flex-grow:有多余空间时 放大多少 默认1

```
flex-shrink:1;空间不足时 缩小多少 默认1, 0 不缩小  
flex-basis: 200px; //设置元素大小 200px
```

Flex 是grow, shrink, basis 三个的简写
如 flex:1 1 250px;

Order :1 元素显示顺序

align-self: flex-end; 元素自身的对齐方式

相对定位和绝对定位

- 理解相对定位和绝对定位
- 相对定位、绝对定位的编码实战

“ 相对定位的元素是相对**自身**进行定位，参照物是**自己**。 ”

“ 绝对定位的元素是相对**离它最近的一个已定位的父级元素**进行定位。 ”

目前可以直接使用 **tar xvJf ***.tar.xz** 来解压

sbt console

scala

React native

Homebrew的安装

Homebrew的安装很简单，只需在终端下输入如下指令：

```
ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

Command+Ctrl+Z

Command+r

**SQLSTATE[HY000] [2002] Connection refused 是因数
据库不充许访问 记得在mamp mysql中勾选意 电脑 访问**

一、Flex 布局是什么？

Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。

任何一个容器都可以指定为 Flex 布局。

```
.box{  
  display: flex;  
}
```

行内元素也可以使用 Flex 布局。

```
.box{  
  display: inline-flex;  
}
```

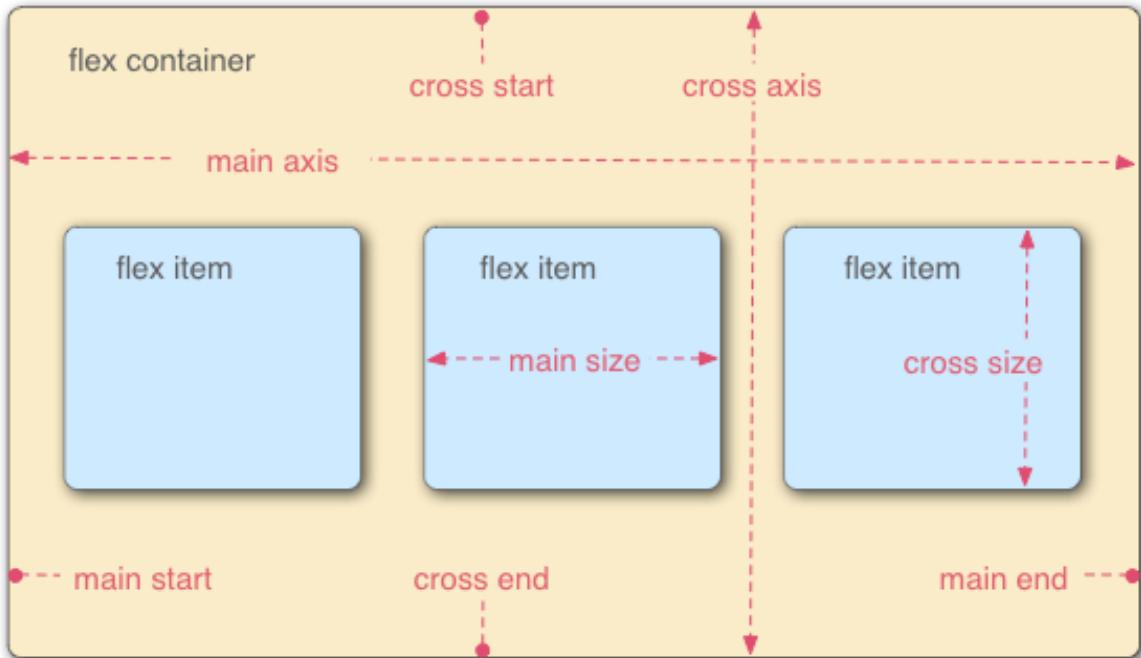
Webkit 内核的浏览器，必须加上-webkit前缀。

```
.box{  
  display: -webkit-flex; /* Safari */  
  display: flex;  
}
```

注意，设为 Flex 布局以后，子元素的float、clear和vertical-align属性将失效。

二、基本概念

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"。它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"。



容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做main start，结束位置叫做main end；交叉轴的开始位置叫做cross start，结束位置叫做cross end。

项目默认沿主轴排列。单个项目占据的主轴空间叫做main size，占据的交叉轴空间叫做cross size。

三、容器的属性

- 以下6个属性设置在容器上。

- flex-direction
- flex-wrap
- flex-flow
- justify-content
- align-items
- align-content

3.1 flex-direction属性

flex-direction属性决定主轴的方向（即项目的排列方向）。

```
.box {
  flex-direction: row | row-reverse | column | column-reverse;
}
```



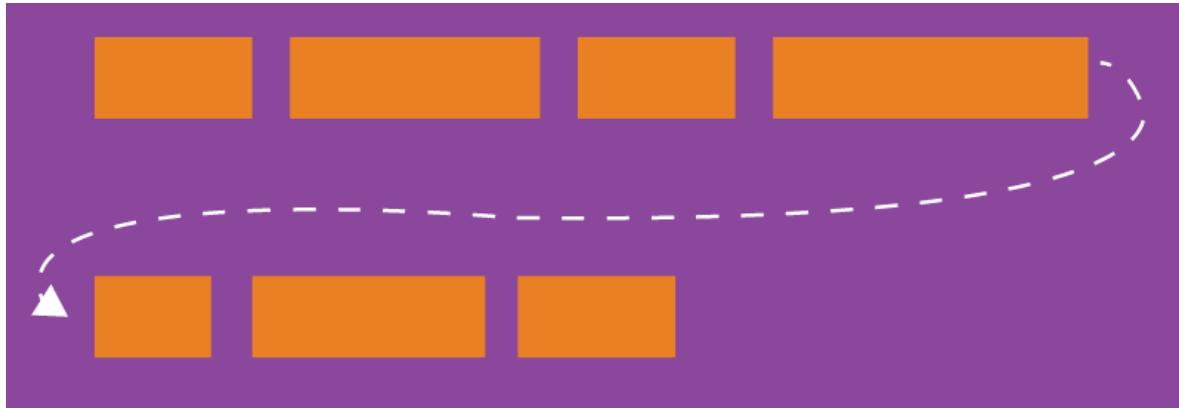
- 它可能有4个值。

row（默认值）：主轴为水平方向，起点在左端。

- row-reverse：主轴为水平方向，起点在右端。
- column：主轴为垂直方向，起点在上沿。
- column-reverse：主轴为垂直方向，起点在下沿。

3.2 flex-wrap属性

默认情况下，项目都排在一条线（又称“轴线”）上。flex-wrap属性定义，如果一条轴线排不下，如何换行。



```
.box{
  flex-wrap: nowrap | wrap | wrap-reverse;
}
```

它可能取三个值。

(1) nowrap (默认) : 不换行。

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|----|

(2) wrap: 换行，第一行在上方。

| | | | | | | | |
|---|----|----|----|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| 9 | 10 | 11 | 12 | | | | |

(3) wrap-reverse: 换行，第一行在下方。

| | | | | | | | |
|---|----|----|----|---|---|---|---|
| 9 | 10 | 11 | 12 | | | | |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

3.3 flex-flow

flex-flow属性是flex-direction属性和flex-wrap属性的简写形式， 默认值为row nowrap。

```
.box {  
  flex-flow: <flex-direction> || <flex-wrap>;  
}
```

3.4 justify-content属性

justify-content属性定义了项目在主轴上的对齐方式。

```
.box {  
  justify-content: flex-start | flex-end | center | space-between | space-around;  
}
```

flex-start



flex-end



center



space-between



space-around



- 它可能取5个值，具体对齐方式与轴的方向有关。下面假设主轴为从左到右。
 - flex-start (默认值) : 左对齐
 - flex-end: 右对齐
 - center: 居中
 - space-between: 两端对齐，项目之间的间隔都相等。
 - space-around: 每个项目两侧的间隔相等。所以，项目之间的间隔比项目与边框的间隔大一倍。

3.5 align-items属性

align-items属性定义项目在交叉轴上如何对齐。

```
.box {  
  align-items: flex-start | flex-end | center | baseline | stretch;
```

}

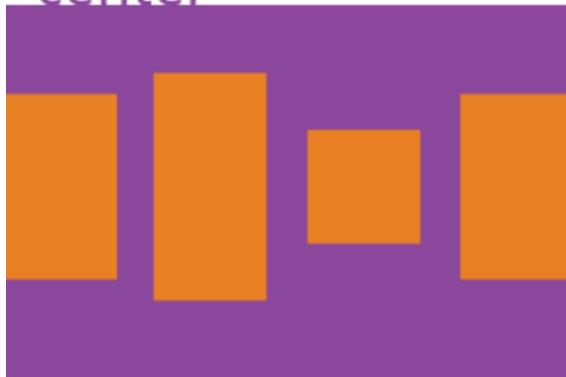
flex-start



flex-end



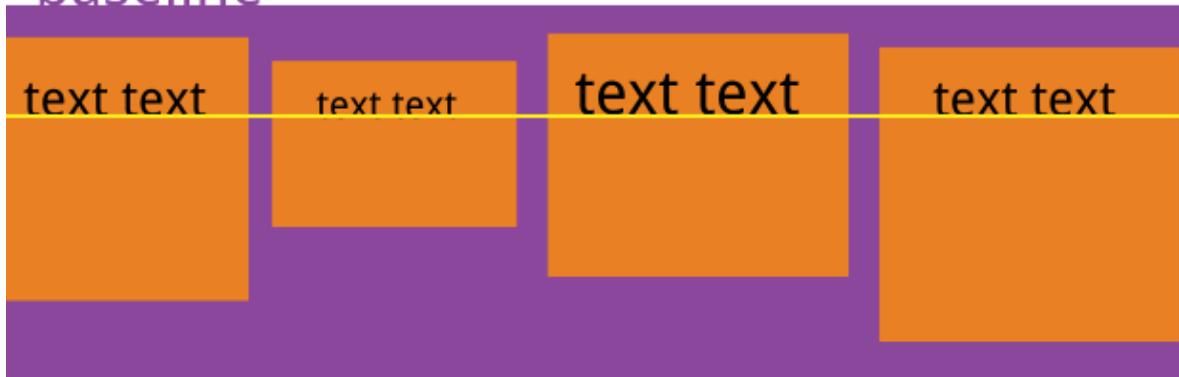
center



stretch



baseline



- 它可能取5个值。具体的对齐方式与交叉轴的方向有关，下面假设交叉轴从上到下。
 - **flex-start**: 交叉轴的起点对齐。
 - **flex-end**: 交叉轴的终点对齐。
 - **center**: 交叉轴的中点对齐。
 - **baseline**: 项目的第一行文字的基线对齐。
 - **stretch** (默认值) : 如果项目未设置高度或设为auto, 将占满整个容器的高度。

3.6 align-content属性

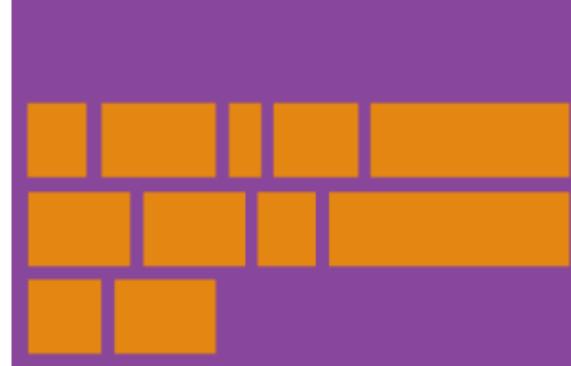
align-content属性定义了多根轴线的对齐方式。如果项目只有一根轴线，该属性不起作用。

```
.box {  
  align-content: flex-start | flex-end | center | space-between | space-around |  
  stretch;  
}
```

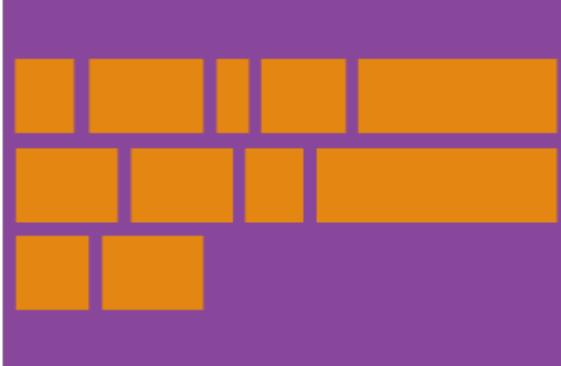
flex-start



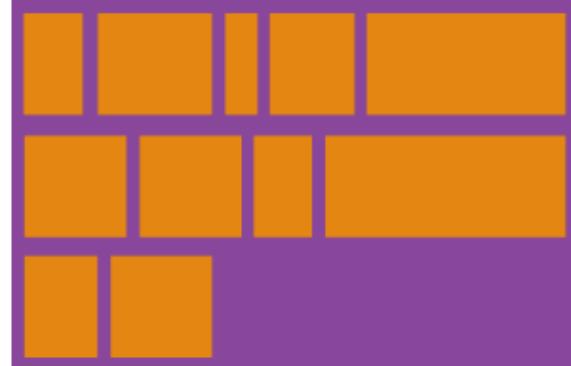
flex-end



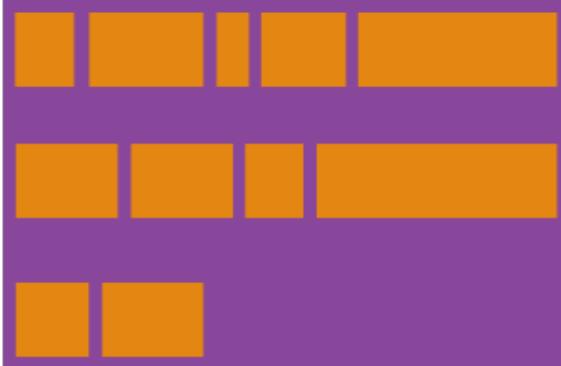
center



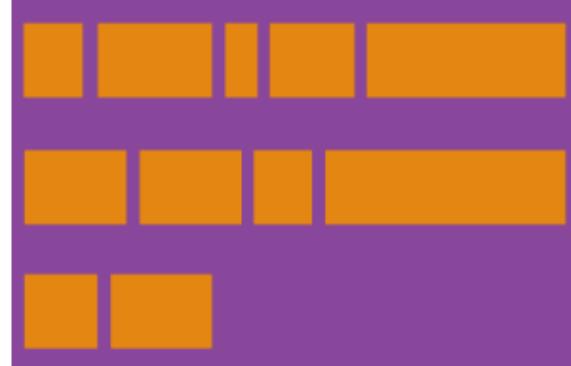
stretch



space-between



space-around



- 该属性可能取6个值。

- flex-start: 与交叉轴的起点对齐。
- flex-end: 与交叉轴的终点对齐。
- center: 与交叉轴的中点对齐。

- space-between: 与交叉轴两端对齐, 轴线之间的间隔平均分布。
- space-around: 每根轴线两侧的间隔都相等。所以, 轴线之间的间隔比轴线与边框的间隔大一倍。
- stretch (默认值) : 轴线占满整个交叉轴。

四、项目的属性

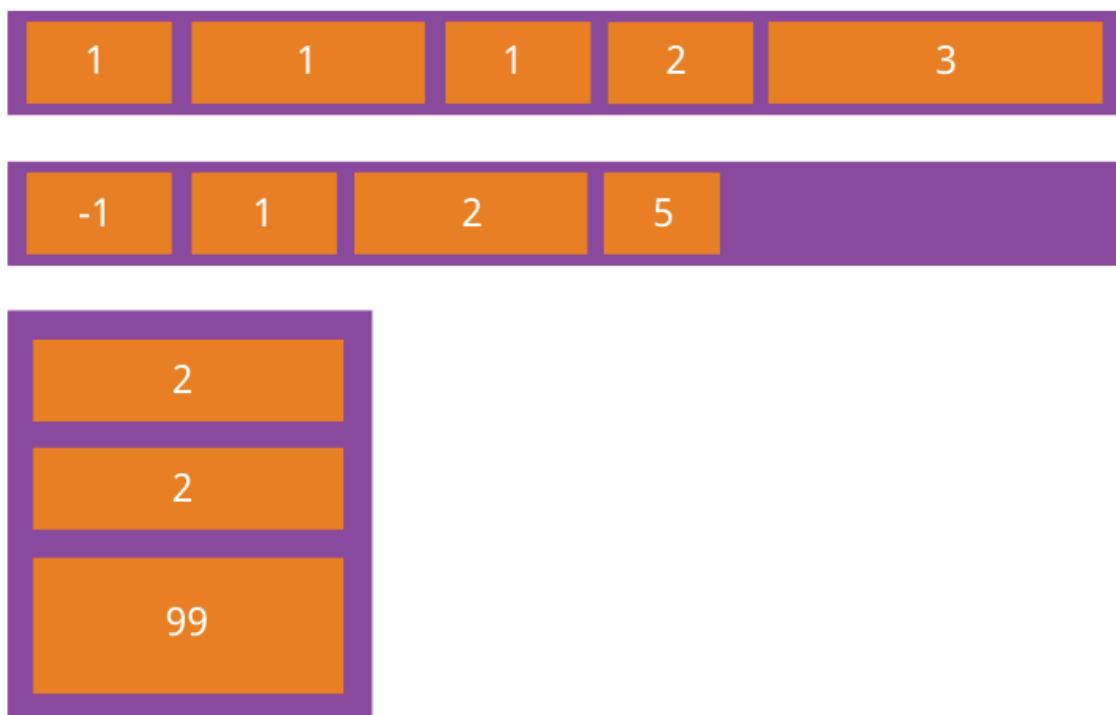
- 以下6个属性设置在项目上。

- order
- flex-grow
- flex-shrink
- flex-basis
- flex
- align-self

4.1 order属性

order属性定义项目的排列顺序。数值越小, 排列越靠前, 默认为0。

```
.item {
  order: <integer>;
}
```



4.2 flex-grow属性

flex-grow属性定义项目的放大比例, 默认为0, 即如果存在剩余空间, 也不放大。

```
.item {
  flex-grow: <number>; /* default 0 */
}
```



如果所有项目的flex-grow属性都为1，则它们将等分剩余空间（如果有的话）。如果一个项目的flex-grow属性为2，其他项目都为1，则前者占据的剩余空间将比其他项多一倍。

4.3 flex-shrink属性

flex-shrink属性定义了项目的缩小比例，默认为1，即如果空间不足，该项目将缩小。

```
.item {  
  flex-shrink: <number>; /* default 1 */  
}
```



如果所有项目的flex-shrink属性都为1，当空间不足时，都将等比例缩小。如果一个项目的flex-shrink属性为0，其他项目都为1，则空间不足时，前者不缩小。
负值对该属性无效。

4.4 flex-basis属性

flex-basis属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为auto，即项目的本来大小。

```
.item {  
  flex-basis: <length> | auto; /* default auto */  
}
```

它可以设为跟width或height属性一样的值（比如350px），则项目将占据固定空间。

4.5 flex属性

flex属性是flex-grow, flex-shrink 和 flex-basis的简写，默认值为0 1 auto。后两个属性可选。

```
.item {  
  flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]  
}
```

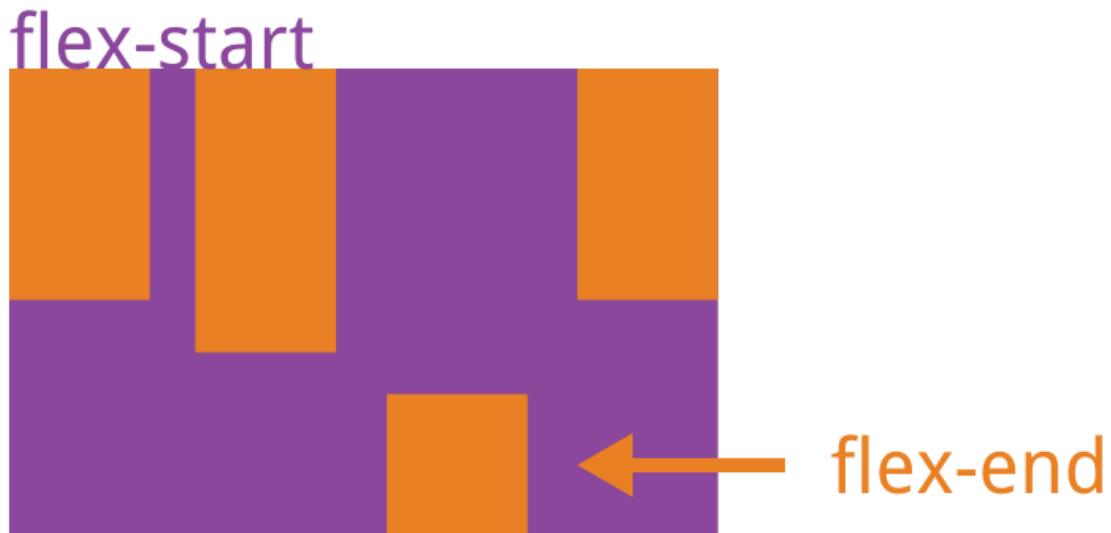
该属性有两个快捷值：auto (1 1 auto) 和 none (0 0 auto)。

建议优先使用这个属性，而不是单独写三个分离的属性，因为浏览器会推算相关值。

4.6 align-self属性

align-self属性允许单个项目有与其他项目不一样的对齐方式，可覆盖align-items属性。默认值为auto，表示继承父元素的align-items属性，如果没有父元素，则等同于stretch。

```
.item {  
  align-self: auto | flex-start | flex-end | center | baseline | stretch;  
}
```



该属性可能取6个值，除了auto，其他都与align-items属性完全一致。

npm install 的时候发生错误

```
npm ERR! code EINTEGRITY  
npm ERR! sha1-W/Rejkm6QYnhfUgnid/RW9FAt7Y= integrity  
chxxxxxxxxxxxxxxxxxxxxDUqxF47jfwOgvK2UM16SEXk=. (10512  
bytes)
```

```
cnpm install webpack webpack-dev-server -g
```

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

1 进入项目目录，运行npm init按照步骤填写最终生成package.json文件

```
2 npm install -g cnpm --registry=https://registry.npm.taobao.org
```

3 已知我们将使用 webpack 作为构建工具，那么就需要安装相应插件，运行 npm install webpack webpack-dev-server --save-dev 来安装两个插件。

```
4 cnpm install --save-dev autoprefixer babel-core babel-loader babel-plugin-react-transform babel-preset-es2015 babel-preset-react css-loader eslint eslint-loader eslint-plugin-react extract-text-webpack-plugin file-loader html-webpack-plugin json-loader koa koa-router less less-loader open-browser-webpack-plugin postcss-loader react-transform-hmr style-loader url-loader
```

```
5 cnpm install babel-preset-env --save-dev
```

```
cnpm install immutable react-addons-css-transition-group react-addons-pure-render-mixin react-redux react-router redux whatwg-fetch --save
cnpm install koa-body --save-dev
cnpm install es6-promise --save

sudo cnpm install babel-preset-es2015 babel-preset-es2015-loose babel-preset-stage-0 babel-eslint babel-loader --save-dev
```

Redux-devTools简单的使用

Installation

```
Sudo cnpm install --save-dev redux-devtools
Sudo cnpm install --save-dev redux-devtools-log-monitor
Sudo cnpm install --save-dev redux-devtools-dock-monitor
```

输入ks.jm.cn/admin 进入后台设置 123

- 1 人员机构模中 —》 人员管理—》 新增 （会自动生成 考试操作员， 密码全部为1,可以到 可以在前台操作员中更改密码）
- 2 试题模块中 可以进行题库 科目管理 （时长， 分值设置， 是否控制考试机器ip地址（在考试ip管理中进行设置）， 是否前台使用（此项必设 才能在前台中有考试项） ）
- 3 试题模块中 可以选择科目 进行题库 查询 或导入
- 4 成绩管理 中可以 查询考生成绩 删除 （导出考生成绩 【在查询出来成绩后 不要进行删除， 这样导出的成绩才会准确】 ）

Git init 创建仓库

Git add filename

git commit -m "test a readme file"

Git status 查看状态

Git commit -am "adasdfadsf". //一次性提交暂存区 并提交仓库

Readme.md

LICENSE

git reset HEAD <file> 把暂存区中的文件恢复 成取消状态

文件在本地修改后,要重新 git add

文件在本地修改后,想要恢复成上次要的文件要 git checkout -- 要恢复的文件名

git checkout -- LICENSE

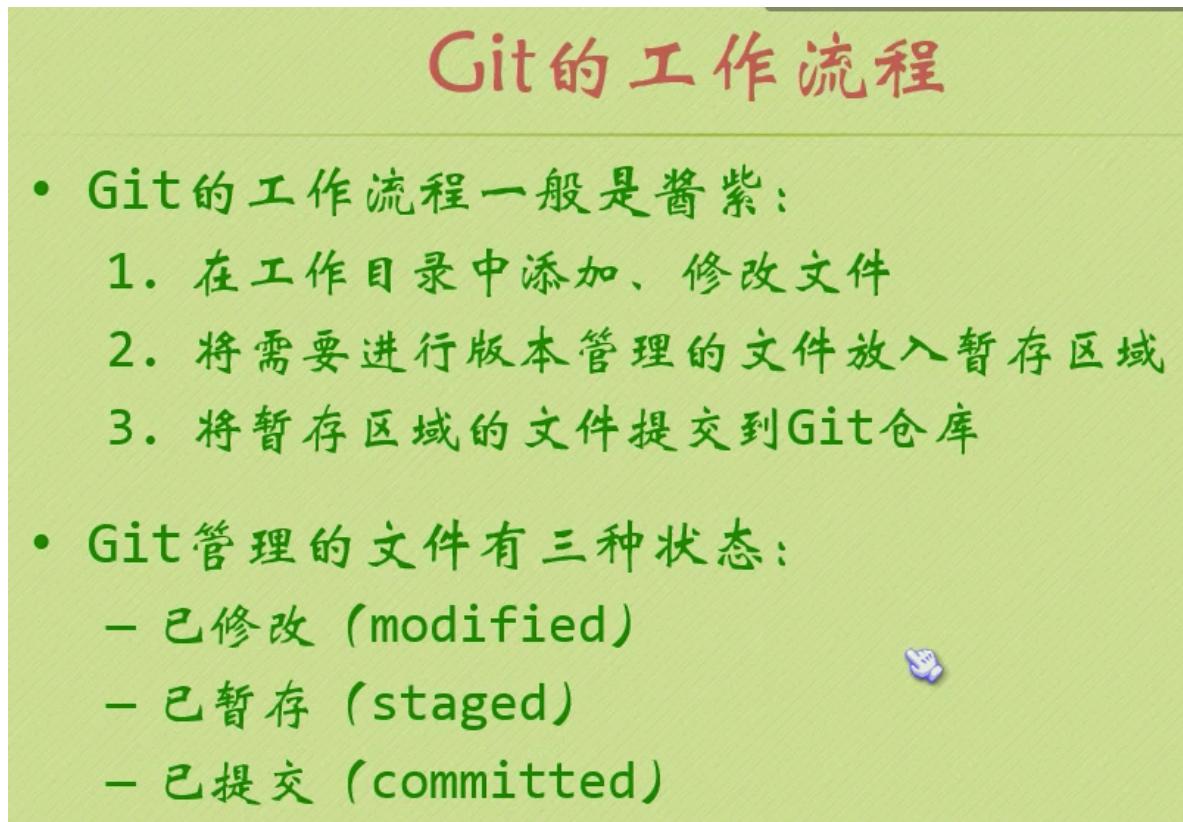
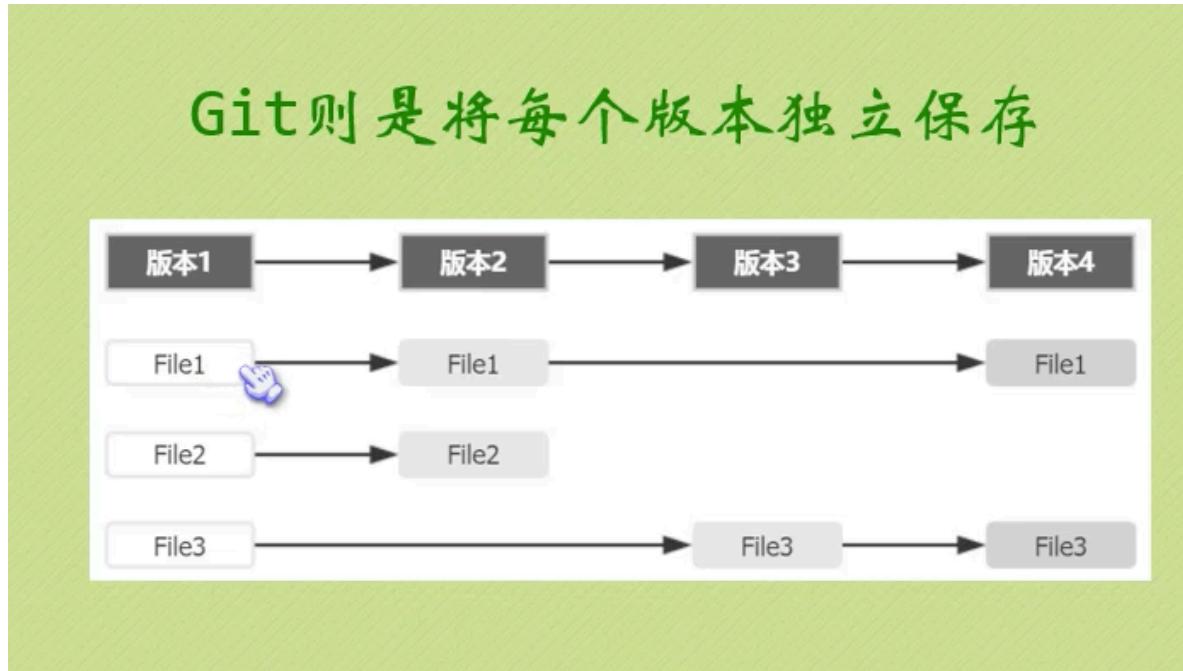
Git log 查看历史提交的 日志

Git diff 快照id1 快照2. 对比

Git diff 快照id1 与当前工作目录中的文件对比

Git diff HEAD 最近一次的快照 与当前工作目录中的文件对比

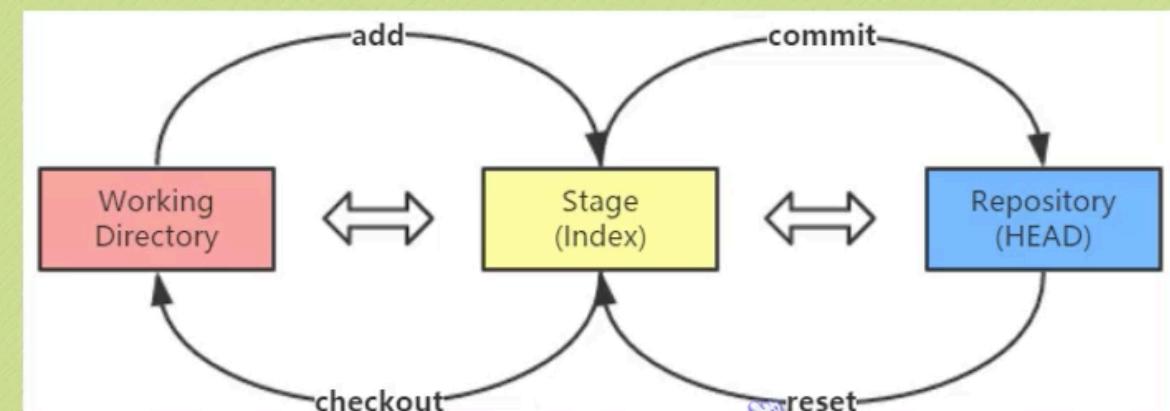
Git diff —CACHED 快照id1 暂存区与 git 仓库最新 对比
git reset HEAD~ 恢复之前的第一个快照到 到暂存区 (state) , 几个波浪线 (~) 就代表恢复到前第几个快照
波浪线 (~) 加数字也可以
如: git reset HEAD~10 代表10个波浪号

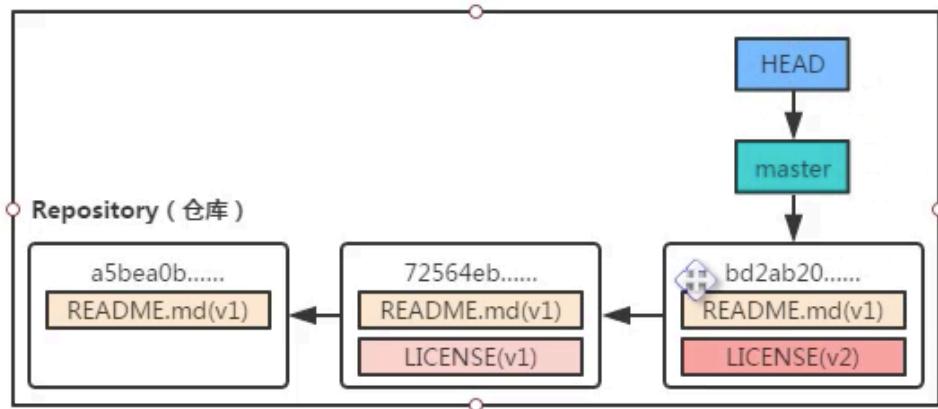


六、

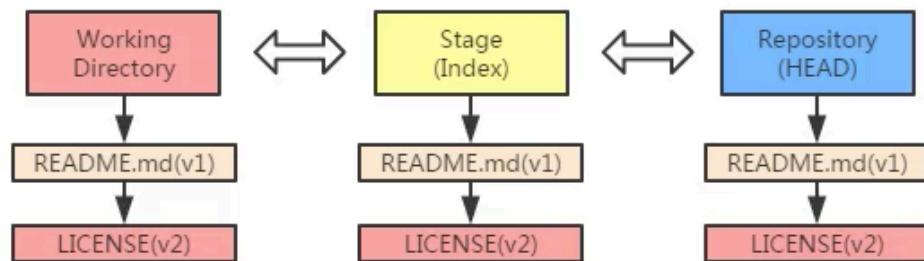
- 将工作目录的文件放到Git仓库只需要两步：
 - git add 文件名
 - git commit -m "你干了啥"

reset和checkout





三棵树



几个命令的功能

- **add**
 - 用于把工作目录的文件放入暂存区域
- **commit**
 - 用于把暂存区域的文件提交到Git仓库
- **reset**
 - 用于把Git仓库的文件还原到暂存区域
- **checkout**
 - 用于把暂存区域的文件还原到工作目录

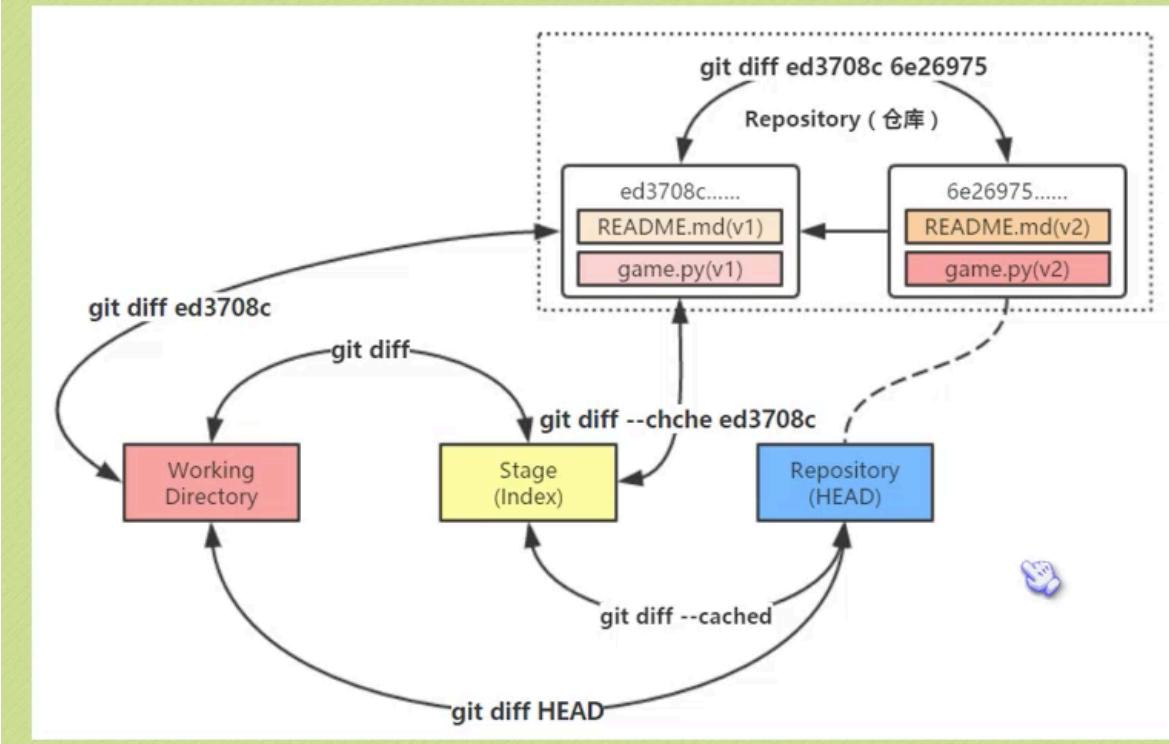
reset命令的选项

- `git rest --mixed HEAD~`
 - 移动HEAD的指向，将其指向上一个快照
 - 将HEAD移动后指向的快照回滚到暂存区域
- `git rest --soft HEAD~`
 - 移动HEAD的指向，将其指向上一个快照
- `git rest --hard HEAD~`
 - 移动HEAD的指向，将其指向上一个快照
 - 将HEAD移动后指向的快照回滚到暂存区域
 - 将暂存区域的文件还原到工作目录

reset命令人回滚快照三部曲

1. 移动 HEAD 的指向 (`--soft`)
2. 将快照回滚到暂存区域 (`--mixed`, 默认)
3. 将暂存区域还原到工作目录 (`--hard`)

终极奥义图



修改最后一次提交

- 在实际开发中，你可能会遇到以下两种情景：
 - 情景一：版本刚一提交（commit）到仓库，突然想起漏掉两个文件还没有添加（add）。
 - 情景二：版本刚一提交（commit）到仓库，突然想起版本说明写得不够全面，无法彰显你本次修改的重大意义.....
- 执行带 `--amend` 选项的 `commit` 提交命令，Git 就会“更正”最近的一次提交。👉

`git commit --amend` 会进行修改说明，使用vi命令来操作
`git checkout -- README.md` 可以恢复删除工作区的文件

删除文件

- `git rm 文件名`

- 该命令删除的只是工作目录和暂存区域的文件，也就是取消跟踪，在下次提交时不纳入版本管理。
- 当工作目录和暂存区域的同一个文件存在不同内容时，执行 `git rm -f 文件名` 命令就可以把两个都删除。
- 如果只删除暂存区域的文件（保留工作目录的），那么你可以执行 `git rm --cached 文件名` 命令实现目的

Git rm -f 文件名 暴力删除文件名 work, state 区不会有提示了(git status)

Git rm —cached 文件名 删除暂存区(state)中的文件名

重命名文件

- `git mv 旧文件名 新文件名`
- `ren/mv 旧文件名 新文件名`
- `git rm 旧文件名`
- `git add 新文件名`

git mv LICENSE license.txt。改名

git commit -m 'rename file'. 改名后记得再次提交

git branch feature 创建分支feature

git log --decorate 查看分支

git checkout feature 切换到分支feature 默认是master分支

git log --decorate --oneline 查看精简分支

git checkout master 切换到master分支

git log --decorate --oneline --graph --all 以图形的方式显示分支

Git merge 分支名 合并分支指定的分支到主分支
git merge feature
git checkout -b feature2 创建并切换到新分支feature2
appledeiMac% git log --decorate --all --oneline --graph
* 2d21331 (**HEAD -> master, feature2**) feature2.txt
* fa1ba28 fix feature
|\
| * 1ae731c (**feature**) feater breach
* | f817d57 change master
|/
* 4a593e2 rename file
* f37f595 11111
* 1198c5f adfasdfasdf
* 8a6d3ae this a tes

appledeiMac% git branch -d feature 删除feature无用的分支
Deleted branch feature (was 1ae731c).
appledeiMac% git branch -d feature2 删除feature2无用的分支
Deleted branch feature2 (was 2d21331).
appledeiMac% git log --decorate --all --oneline --graph
* 2d21331 (**HEAD -> master**) feature2.txt
* fa1ba28 fix feature
|\
| * 1ae731c feater breach
* | f817d57 change master
|/
* 4a593e2 rename file
* f37f595 11111
* 1198c5f adfasdfasdf
* 8a6d3ae this a test
appledeiMac%

git checkout HEAD~ 返回上一个快照 并创建了一个匿名分支
在匿名分支所做的操作都会被丢掉，所以可以在匿名分支做测试
appledeiMac% git log --decorate --oneline --graph --all
* 7c9827d (**HEAD**) 4.txt
| * f359ff6 (**master**) 3.txt
|/
* 2e67d9f 2.txt
* e927832 1.txt
appledeiMac% git checkout master
Warning: you are leaving 1 commit behind, not connected to
any of your branches:

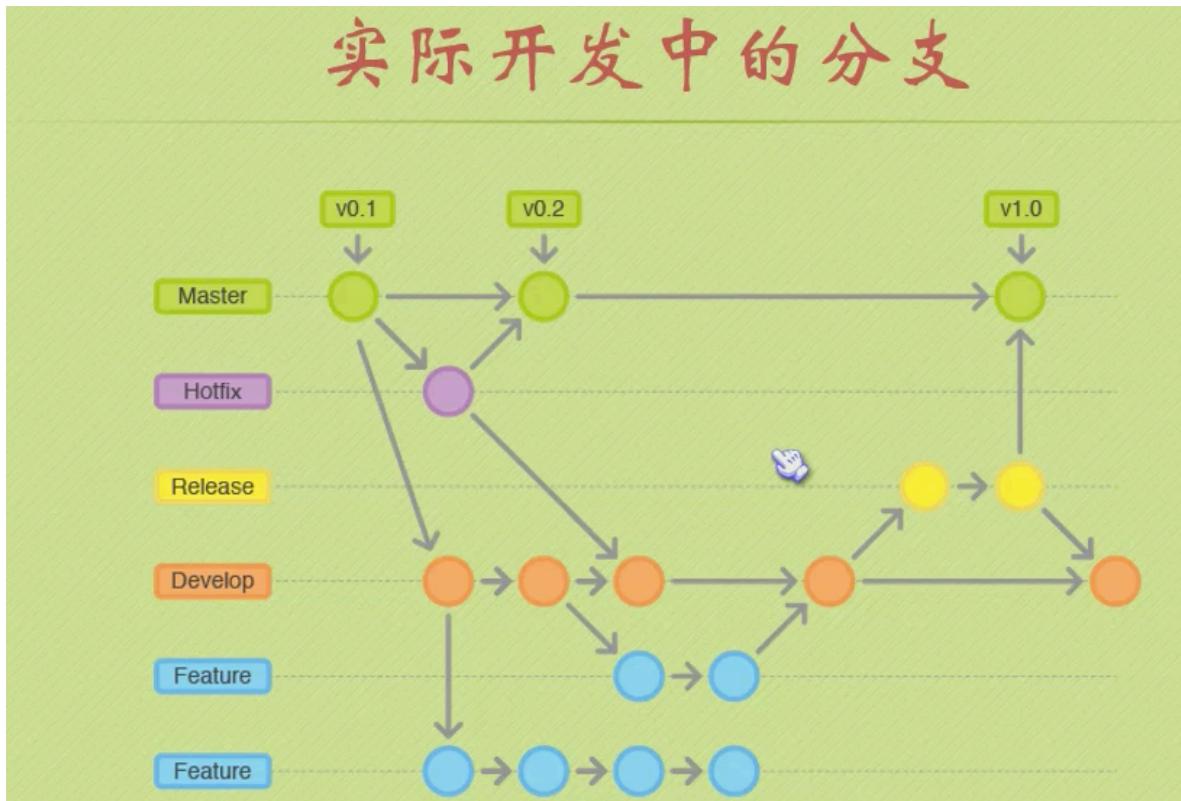
7c9827d 4.txt

If you want to keep it by creating a new branch, this may be a good time to do so with:

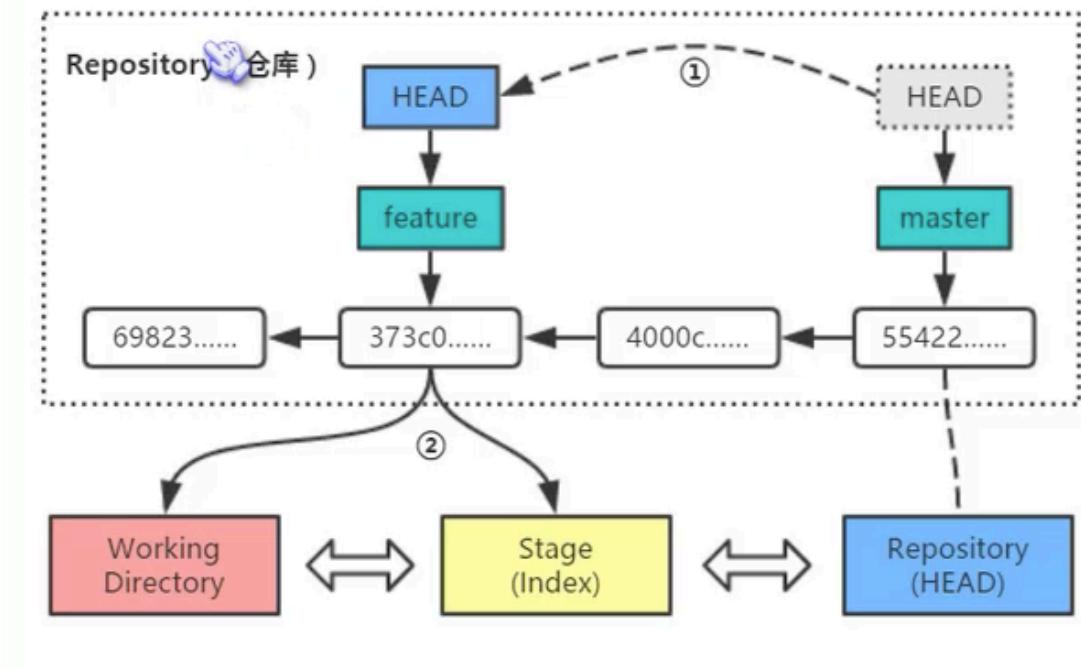
git branch <new-branch-name> 7c9827d (可以现在创建一个分支，把I带上就不会丢失所做的工作)

Switched to branch 'master'

如果 checkout 不带id 它会把暂存区的快照 放回工作区 如 git checkout HEAD~命令



所以执行 `git checkout 373c0` 命令，Git 主要就是做了下边这两件事（当然事实上 Git 还做了更多）：



恢复文件

`checkout` 命令和 `reset` 命令都可以用于恢复指定快照的指定文件，并且它们都不会改变 `HEAD` 指针的指向。

下面开始划重点：

它们的区别是 `reset` 命令只将指定文件恢复到暂存区域（`--mixed`），而 `checkout` 命令是同时覆盖暂存区域和工作目录。

注意：也许你试图使用 `git reset --hard HEAD~ README.md` 命令让 `reset` 同时覆盖工作目录，但 Git 会告诉你这是徒劳（此时 `reset` 不允许使用 `--soft` 或 `--hard` 选项）。

这样看来，在恢复文件方面，`reset` 命令要比 `checkout` 命令更安全一些。

恢复快照

`reset` 命令是用来“回到过去”的，根据选项的不同，`reset` 命令将移动 `HEAD` 指针（`--soft`）-> 覆盖暂存区域（`--mixed`, 默认）-> 覆盖工作目录（`--hard`）。

`checkout` 命令虽说是用于切换分支，但前面你也看到了，它事实上也是通过移动 `HEAD` 指针和覆盖暂存区域、工作目录来实现的。

那问题来了：它们有什么区别呢？

下面开始划重点：

第一个区别是，对于 `reset --hard` 命令来说，`checkout` 命令更安全。因为 `checkout` 命令在切换分支前会先检查一下当前的工作状态，如果不是“clean”的话，Git 不会允许你这样做；而 `reset --hard` 命令则是直接覆盖所有数据。

另一个区别是如何更新 `HEAD` 指向，`reset` 命令会移动 `HEAD` 所在分支的指向，而 `checkout` 命令只会移动 `HEAD` 自身来指向另一个分支。

```
git clone https://github.com/class0123456789/GitTest.git  
Cd GitTest  
git add 1.txt  
git commit -m 'test first code'  
git push -u origin master 首次上传GitHub时用到 GitHub  
Git status  
git push origin master 再次上传GitHub时用到  
git push 上传整个项目到GitHub
```

在GitHub上的这个learngit仓库还是空的,GitHub告诉我们,可以从这个仓库克隆出新的。也可以把一个已有的本地仓库与之关联,然后,把本地仓库的内容推送到GitHub仓库。

```
git add .  
git remote add origin git@git.oschina.net:cherrywy/gerenjianli.git  
git remote rm origin  
把上面的git@git.oschina.net:cherrywy/gerenjianli.git替换成你自己的GitHub的  
ssh,
```

加后,远程库的名字就是origin,这是Git默认的叫法,也可以改成别的,但是origin这个名字是固定的。
git push -u origin master
出错了解决办法
git pull --rebase
git push

下一步,就可以把本地库的所有内容推送到远程库上:

```
git push -u origin master  
又出错了  
git pull --rebase origin master
```

最后集成代码

```
cd g:\cd G:\DWworkingspace\gerenjianli  
git add . git remote add origin git@git.oschina.net:cherrywy/gerenjianli.git  
git remote rm origin
```

```
git add .  
git remote add origin git@git.oschina.net:cherrywy/gerenjianli.git  
git push -u origin master  
git pull --rebase  
git push  
git push -u origin master  
git pull --rebase origin master
```

```
git add .
git commit -m "This is a new commit for what I originally planned to be
amended"
git push origin master
```

Mac下使用SSH(密钥)访问Github

如你本机新建git项目 地址: `git@github.com:yourName/yourRepo.git`, 远程代码库服务器地址: 192.168.10.1, 远程代码服务器账户名密码: admin 密码: 123456

1.首先确保配置正确

a. 创建本地ssh-key

```
$ ssh-keygen -t rsa
```

在`~/.ssh/`下会生成 `id_rsa.pub`密钥文件

b.上传密钥文件 `id_rsa.pub`到代码服务器

```
$ cat /home/XXXX/.ssh/id_rsa.pub | ssh admin@192.168.10.1
```

```
"cat >> ~/.ssh/authorized_keys"
```

会提示输入如代码服务器密码, 输入密码: 123456即可。

c.添加全局变量:

```
$ git config --global user.name 'yourName'
```

```
$ git config --global user.email 'yourEmail@yourEmail.com'
```

2.进入要上传的git项目

```
$ git remote add origin git@github.com:yourName/yourRepo.git
```

之后进入`.git`, 打开`config`, 这里会多出一个remote “origin”内容, 这就是刚才添加的远程地址, 也可以直接修改`config`来配置远程地址。

3.提交、上传

a.接下来在本地仓库里添加一些文件, 比如`README`,

```
$ git add README
```

```
$ git commit -m "first commit"
```

b.上传到代码服务器:

```
$ git push origin master
```

`git push`命令会将本地仓库推送到远程服务器。

`git pull`命令则相反。

修改完代码后, 使用`git status`可以查看文件的差别, 使用`git add`添加要commit的文件, 也可以用`git add -i`来智能添加文件。之后`git commit`提交本次修改, `git push`上传到代码服务器。

我的github上传步骤:

```
git clone https://github.com/class0123456789/es6study.git
```

```
cd es6study
```

```
cp -r ../../gites6study/* .
```

```
git add .
```

```
git commit -m 'upload all study es6 code'
```

```
git push
```

Nginx 代理

```
/usr/local/etc/nginx/servers/node-com-3000.conf
```

```
upstream imooc {  
    server 127.0.0.1:3000;  
}  
  
server {  
    listen 8088;  
    server_name localhost;  
  
    location / {  
        proxy_set_header X-Real-IP $remote_addr;  
        proxy_set_header X-Forward-For $proxy_add_x_forwarded_for;  
        proxy_set_header Host $http_host;  
        proxy_set_header X-Nginx-Proxy true;  
        proxy_pass http://imooc;  
        proxy_redirect off;  
    }  
}
```

```
sudo nginx -t 测试配置文件是否正确
```

```
sudo nginx 启动
```

```
sudo nginx -s stop 停止
```

```
sudo nginx -s reload 重启
```

命令:**express -e ./ express** 表示安装**express -e** 表示使用**ejs**作为模板 **./**表示当前目录中 (使用上面的命令之前我们应该使用**npm**安装**express**框架)

```
sudo cnpm install -g express
```

```
sudo cnpm install -g express-generator
```

```
express -e . 建立express框架 环境
```

```
cnpm install (package.json)
```

用webpack-dev-server开发时代理，解决开发时跨域问题

```
npm install webapck webpack-dev-server --save-dev
```

```
new OpenBrowserPlugin({
  url: 'http://localhost:9000'
}),

// 提供公共代码
//new webpack.optimize.CommonsChunkPlugin({
//  name: 'vendor',
//  filename: '/js/[name].js'
//}),

// 可在业务 js 代码中使用 __DEV__ 判断是否是dev模式
new webpack.DefinePlugin({
  __DEV__: JSON.stringify(JSON.parse((process.env.NODE_ENV || 'development') === 'development'))
}),
devServer: {
  //colors: true, //终端中输出结果为彩色
  port : 9000,
  historyApiFallback: true, //不跳转，在开发单页应用时非常有用
  inline: true, //实时刷新
  hot: true , // 使用热加载插件 HotModuleReplacementPlugin
  proxy: {
    '/get': {
      target: 'http://localhost:3000',
      pathRewrite: {'^/column' : '/column'},
      changeOrigin: true
    }
  }
}
```

nginx反向代理-解决前端跨域问题

1.定义

跨域是指a页面想获取b页面资源，如果a、b页面的协议、域名、端口、子域名不同，所进行的访问行动都是跨域的，而浏览器为了安全问题一般都限制了跨域访问，也就是不允许跨域请求资源。注意：跨域限制访问，其实是浏览器的限制。理解这一点很重要！！！

2.跨域访问示例

假设有两个网站，A网站部署在：http://localhost:81 即本地ip端口81上；B网站部署在：http://localhost:82 即本地ip端口82上。

现在A网站的页面想去访问B网站的信息，A网站页面的代码如下（这里使用jquery的异步请求）：

```
<h2>Index</h2>
<div id="show"></div>
<script type="text/javascript">
$(function () {
    $.get("http://localhost:82/api/values", {}, function (result)
{
    $("#show").html(result);
})
})
```

3.nginx反向代理解决跨域问题

3.1nginx配置

关于nginx的配置可以查看另一篇博文：<http://www.cnblogs.com/renjing/p/6126284.html>。找到nginx的配置文件“nginx.conf”，修改一下信息

```
server {
    listen 80; #监听80端口，可以改成其他端口
    server_name localhost; # 当前服务的域名

    #charset koi8-r;

    #access_log logs/host.access.log main;

    location / {
        proxy_pass http://localhost:81;
        proxy_redirect default;
    }

    location /apis { #添加访问目录为/apis的代理配置
        rewrite ^/apis/(.*)$ /$1 break;
        proxy_pass http://localhost:82;
    }
}

#以下配置省略
```

配置解释：

1.由配置信息可知，我们让nginx监听localhost的80端口，网站A与网站B的访问都是经过localhost的80端口进行访问。

2.我们特殊配置了一个“/apis”目录的访问，并且对url执行了重写，最后使以“/”

apis”开头的地址都转到“http://localhost:82”进行处理。

3.rewrite ^/apis/(.*)\$ /\$1 break;

代表重写拦截进来的请求，并且只能对域名后边以“/apis”开头的起作用，例如
www.a.com/apis/msg?x=1重写。只对/apis重写。

rewrite后面的参数是一个简单的正则 ^/apis/(.*)\$, \$1代表正则中的第一个(), \$2代表第二个() 的值,以此类推。

break代表匹配一个之后停止匹配。

3.2 访问地址修改

既然配置了nginx，那么所有的访问都要走nginx，而不是走网站原本的地址（A网站localhost:81,B网站localhost:82）。所以要修改A网站中的ajax访问地址，把访问地址由

“http://localhost:82/api/values”改成» » » “/apis/api/values”。如下代码

5.总结

浏览器跨域的解决方式有很多种：

1.jsonp 需要目标服务器配合一个callback函数。

2.window.name+iframe 需要目标服务器响应window.name。

3.window.location.hash+iframe 同样需要目标服务器作处理。

4.html5的 postMessage+ifrme 这个也是需要目标服务器或者说是目标页面写一个postMessage，主要侧重于前端通讯。

5.CORS 需要服务器设置header： Access-Control-Allow-Origin。

6.nginx反向代理 这个方法一般很少有人提及，但是他可以不用目标服务器配合，不过需要你搭建一个中转nginx服务器，用于转发请求。

个人觉得6才是正规的解决方案

EX6学习

```
{  
    // #构造函数#  
    let regex = new RegExp('xyz', 'i'); //第一个参数是字符串，第二个是修饰符  
    let regex2 = new RegExp(/xyz/i); //第一个参数是正则表达式，不接受第二个参数，否则会报错  
    console.log(regex.test('xyz123'), regex2.test('xyz123'));  
    console.log(regex.test('xyZ123'), regex2.test('xyZ123'));
```

```
    let regex3 = new RegExp(/abc/ig, 'i');  
    console.log(regex3.flags); //原有正则对象的修饰符是ig，它会被第二个参数i覆盖
```

```
}
```

// 字符串对象的4个使用正则表达式的方法：match(), replace(), search(), split()这四个方法全部调用RegExp的实例的方法。

```
{
```

```

let regex = new RegExp('xyz', 'ig');
console.log(regex.test('xyz0XYZ1xyz2'), regex.exec('xyz0XYZ1xyz2'));
}

{
  // y修饰符
  let s = 'bbbb_bbb_bb_b';
  var a1 = /b+/g;
  var a2 = /b+/y;    //es6

  console.log(a1.exec(s), a2.exec(s)); // ["bbbb"],["bbbb"]
  console.log(a1.exec(s), a2.exec(s)); // ["bbb"],null

  console.log(a1.sticky, a2.sticky); //ES6 sticky表示是否开启了粘连模式
}

{
  //u修饰符 就是处理unicode
  console.log('u修饰符',/^uD83D/.test('\uD83D\uDC2A')); // true 不加u 把当作2个字符
  console.log('u修饰符',/^uD83D/u.test('\uD83D\uDC2A')); // false 加u 把当作1个字符
  // 大括号表示Unicode字符， 只有加上u才能识别
  console.log(/\u{61}/.test('a')); // false
  console.log(/\u{61}/u.test('a')); // true
  console.log(`\u{20BB7}`) //es6中``模板
  console.log(/\u{20BB7}`/u.test('吉')); // true
  // 点(.) 字符不能识别码点大于0xFFFF的Unicode字符， 必须加上u修饰符。
  let s = '吉';
  console.log('大于0xFFFF的Unicode字符',/^$/u.test(s)); // false
  console.log('使用u字符',/^$/u.test(s)); // true

  // 使用u修饰符后， 所有量词都会正确识别大于码点大于0xFFFF的Unicode字符。
  console.log('量词',/a{2}/.test('aa')); // true
  console.log('量词',/a{2}/u.test('aa')); // true
  console.log('量词',/吉{2}/.test('吉吉')); // false
  console.log('量词',/吉{2}/u.test('吉吉')); // true
}

{
  // #正则表达式中， 点(.) 是一个特殊字符， 代表任意的单个字符， 但是行终止符（line terminator character）除外
  // U+000A 换行符 (\n)
  // U+000D 回车符 (\r)
  // U+2028 行分隔符 (line separator)
}

```

```
// U+2029 段分隔符 (paragraph separator)
// 只是一个提案目前还不支持
// let reg=/test.go/s;
// console.log(reg.test('test\ngo'));
// console.log(reg.test('test\ngo'));
console.log('s变通方法',/foo.bar/.test('foo\nbar'));
console.log('s变通方法',/foo[^]bar/.test('foo\nbar'));
}
```

```
npm install -g pm2
brew install mongodb
```

=> **Caveats**

To have launchd start mongodb now and restart at login:

```
brew services start mongodb
```

Or, if you don't want/need a background service you can just run:

```
mongod --config /usr/local/etc/mongod.conf
```

=> **Summary**

 /usr/local/Cellar/mongodb/3.4.9: 19 files, 284.9MB

```
appledeiMac% brew services start mongodb
```

=> **Tapping homebrew/services**

```
Cloning into '/usr/local/Homebrew/Library/Taps/homebrew/homebrew-
services'...
```

```
remote: Counting objects: 12, done.
```

```
remote: Compressing objects: 100% (8/8), done.
```

```
remote: Total 12 (delta 0), reused 7 (delta 0), pack-reused 0
```

```
Unpacking objects: 100% (12/12), done.
```

```
Tapped 0 formulae (40 files, 54.2KB)
```

=> **Successfully started `mongodb` (label: homebrew.mxcl.mongodb)**

```
appledeiMac% brew services restart mongodb
```

```
Stopping `mongodb` ... (might take a while)
```

=> **Successfully stopped `mongodb` (label: homebrew.mxcl.mongodb)**

=> **Successfully started `mongodb` (label: homebrew.mxcl.mongodb)**

```
appledeiMac% brew services stop mongodb
```

```
Stopping `mongodb` ... (might take a while)
```

=> **Successfully stopped `mongodb` (label: homebrew.mxcl.mongodb)**

mongod 启动服务端以后启动就可以直接输入**mongod** 启动了
mongo 重新打开一个终端 ,进入管理数据库管理操作

9. 出现上图的提示就代表连接成功了， 终端上会一直显示一个 ‘>’ 符号， 此时就可以输入**mongodb**的sql命令 了(这里我列出一些基本的):

```
*test 是我建的一个集合名字
```

```
show dbs //显示数据库
```

```
use test //使用某个数据库
```

```
db.test.insert({'name':'byc'}) //插入一条记录
db.test.find() //查找所有记录
db.test.findOne() //查找一条记录
db.dropDatabase() //删除数据库
db.test.drop //删除指定集合
show collections //显示所有集合
db.createCollection('byc') //创建集合
db.test.save({}) //插入记录db.test.update({_id:1},{$set:{name:'test',age:20}})
db.test.remove({}) //删除所有集合
for(var i=1;i<=10;i++){db.test.insert({"name":"king"+i,"age":i})} //循环插入10条记录
db.test.find().pretty() //格式化显示查询结果
db.test.find().count() //查询数据条数
db.test.find({"age":5}) //查找age是5的条目
db.test.find({"age":{$gt:5}}) //查找age大于5的条目
db.test.find({"age":{$gt:5}}).sort({"age":1}) //查找age大于5的条目且升序排列
db.test.find({"age":{$gt:5}}).sort({"age":-1}) //查找age大于5的条目且降序排列
```

10. 可以安装个图形化工具进行连接操作，我这里用了一个**mongobooster** 的软件，

11. 要停止**mongodb**一定要正确的退出,不然下次再次连接数据库会出现问题.

```
use admin;
db.shutdownServer();
```

MONGODB基本命令用

成功启动MongoDB后，再打开一个命令行窗口输入mongo，就可以进行数据库的一些操作。

输入help可以看到基本操作命令：

show dbs:显示数据库列表

show collections: 显示当前数据库中的集合（类似关系数据库中的表）

show users: 显示用户

use <db name>: 切换当前数据库，这和MS-SQL里面的意思一样

db.help(): 显示数据库操作命令，里面有很多的命令

db.foo.help(): 显示集合操作命令，同样有很多的命令，foo指的是当前数据库下，一个叫foo的集合，并非真正意义上的命令

db.foo.find(): 对于当前数据库中的foo集合进行数据查找（由于没有条件，会列出所有数据）

db.foo.find({ a : 1 }): 对于当前数据库中的foo集合进行查找，条件是数据中有一个属性叫a，且a的值为1

MongoDB没有创建数据库的命令，但有类似的命令。

如：如果你想创建一个“myTest”的数据库，先运行use myTest命令，之后就做一些操作（如：db.createCollection('user')），这样就可以创建一个名叫“myTest”的数据库。

数据库常用命令

1、Help查看命令提示

```
help
db.help();
db.yourColl.help();
db.youColl.find().help();
rs.help();

2、切换/创建数据库
use yourDB; 当创建一个集合(table)的时候会自动创建当前数据库

3、查询所有数据库
show dbs;

4、删除当前使用数据库
db.dropDatabase();

5、从指定主机上克隆数据库
db.cloneDatabase("127.0.0.1"); 将指定机器上的数据库的数据克隆到当前数据库

6、从指定的机器上复制指定数据库数据到某个数据库
db.copyDatabase("mydb", "temp", "127.0.0.1"); 将本机的mydb的数据复制到temp
数据库中

7、修复当前数据库
db.repairDatabase();

8、查看当前使用的数据库
db.getName();
db; db和getName方法是一样的效果，都可以查询当前使用的数据库

9、显示当前db状态
db.stats();

10、当前db版本
db.version();

11、查看当前db的链接机器地址
db.getMongo();

Collection聚集集合
1、创建一个聚集集合 (table)
db.createCollection("collName", {size: 20, capped: 5, max: 100});

2、得到指定名称的聚集集合 (table)
db.getCollection("account");

3、得到当前db的所有聚集集合
db.getCollectionNames();

4、显示当前db所有聚集索引的状态
db.printCollectionStats();

用户相关
1、添加一个用户
db.addUser("name");
db.addUser("userName", "pwd123", true); 添加用户、设置密码、是否只读

2、数据库认证、安全模式
db.auth("userName", "123123");

3、显示当前所有用户
show users;

4、删除用户
db.removeUser("userName");
```

其他

1、查询之前的错误信息

```
db.getPrevError();
```

2、清除错误记录

```
db.resetError();
```

查看聚集集合基本信息

1、查看帮助 db.yourColl.help();

2、查询当前集合的数据条数 db.yourColl.count();

3、查看数据空间大小 db.userInfo.dataSize();

4、得到当前聚集集合所在的db db.userInfo.getDB();

5、得到当前聚集的状态 db.userInfo.stats();

6、得到聚集集合总大小 db.userInfo.totalSize();

7、聚集集合储存空间大小 db.userInfo.storageSize();

8、Shard版本信息 db.userInfo.getShardVersion()

9、聚集集合重命名 db.userInfo.renameCollection("users"); 将userInfo重命名为 users

10、删除当前聚集集合 db.userInfo.drop();

聚集集合查询

1、查询所有记录

```
db.userInfo.find();
```

相当于： select* from userInfo;

默认每页显示20条记录，当显示不下的情况下，可以用it迭代命令查询下一页数据。

注意：键入it命令不能带“；”

但是你可以设置每页显示数据的大小，用DBQuery.shellBatchSize= 50;这样每页就显示50条记录了。

2、查询去掉后的当前聚集集合中的某列的重复数据

```
db.userInfo.distinct("name");
```

会过滤掉name中的相同数据

相当于： select distinct name from userInfo;

3、查询age = 22的记录

```
db.userInfo.find({"age": 22});
```

相当于： select * from userInfo where age = 22;

4、查询age > 22的记录

```
db.userInfo.find({age: {$gt: 22}});
```

相当于： select * from userInfo where age > 22;

5、查询age < 22的记录

```
db.userInfo.find({age: {$lt: 22}});
```

相当于： select * from userInfo where age < 22;

6、查询age >= 25的记录

```
db.userInfo.find({age: {$gte: 25}});
```

相当于： select * from userInfo where age >= 25;

7、查询age <= 25的记录

```
db.userInfo.find({age: {$lte: 25}});
```

8、查询age >= 23 并且 age <= 26

```
db.userInfo.find({age: {$gte: 23, $lte: 26}});
```

9、查询name中包含 mongo的数据

```
db.userInfo.find({name: /mongo/});
```

//相当于%%

```
select * from userInfo where name like '%mongo%';
```

10、查询name中以mongo开头的

```
db.userInfo.find({name: /^mongo/});
```

```
select * from userInfo where name like 'mongo%';
```

11、查询指定列name、 age数据

```
db.userInfo.find({}, {name: 1, age: 1});
```

相当于： select name, age from userInfo;

当然name也可以用true或false,当用ture的情况下和name:1效果一样，如果用false就是排除name，显示name以外的列信息。

12、查询指定列name、 age数据, age > 25

```
db.userInfo.find({age: {$gt: 25}}, {name: 1, age: 1});
```

相当于： select name, age from userInfo where age >25;

13、按照年龄排序

升序： db.userInfo.find().sort({age: 1});

降序： db.userInfo.find().sort({age: -1});

14、查询name = zhangsan, age = 22的数据

```
db.userInfo.find({name: 'zhangsan', age: 22});
```

相当于： select * from userInfo where name = ‘zhangsan’ and age = ‘22’;

15、查询前5条数据

```
db.userInfo.find().limit(5);
```

相当于： selecttop 5 * from userInfo;

16、查询10条以后的数据

```
db.userInfo.find().skip(10);
```

相当于： select * from userInfo where id not in (

```
selecttop 10 * from userInfo
```

```
);
```

17、查询在5-10之间的数据

```
db.userInfo.find().limit(10).skip(5);
```

可用于分页， limit是pageSize, skip是第几页*pageSize

18、or与查询

```
db.userInfo.find({$or: [{age: 22}, {age: 25}]});  
相当于: select * from userInfo where age = 22 or age = 25;
```

19、查询第一条数据

```
db.userInfo.findOne();  
相当于: select top 1 * from userInfo;  
db.userInfo.find().limit(1);
```

20、查询某个结果集的记录条数

```
db.userInfo.find({age: {$gte: 25}}).count();  
相当于: select count(*) from userInfo where age >= 20;  
如果要返回限制之后的记录数量，要使用count(true)或者count(非0)  
db.users.find().skip(10).limit(5).count(true);
```

21、按照某列进行排序

```
db.userInfo.find({sex: {$exists: true}}).count();  
相当于: select count(sex) from userInfo;  
索引
```

1、创建索引

```
db.userInfo.ensureIndex({name: 1});  
db.userInfo.ensureIndex({name: 1, ts: -1});
```

2、查询当前聚集集合所有索引

```
db.userInfo.getIndexes();
```

3、查看总索引记录大小

```
db.userInfo.totalIndexSize();
```

4、读取当前集合的所有index信息

```
db.users.reIndex();
```

5、删除指定索引

```
db.users.dropIndex("name_1");
```

6、删除所有索引索引

```
db.users.dropIndexes();
```

修改、添加、删除集合数据

1、添加

```
db.users.save({name: 'zhangsan', age: 25, sex: true});  
添加的数据的数据列，没有固定，根据添加的数据为准
```

2、修改

```
db.collection.update(criteria, objNew, upsert, multi )
```

criteria:update的查询条件，类似sql update查询内where后面的
objNew:update的对象和一些更新的操作符（如\$,\$inc...）等，也可以理解为sql
update查询内set后面的。

upsert : 如果不存在update的记录，是否插入objNew,true为插入，默认是false，
不插入。

multi : mongodb默认是false,只更新找到的第一条记录，如果这个参数为true,就把
按条件查出来多条记录全部更新。

```
db.users.update({age: 25}, {$set: {name: 'changeName'}}, false, true);  
相当于: update users set name = 'changeName' where age = 25;
```

```
db.users.update({name: 'Lisi'}, {$inc: {age: 50}}, false, true);  
相当于: update users set age = age + 50 where name = 'Lisi';
```

```
db.users.update({name: 'Lisi'}, {$inc: {age: 50}, $set: {name: 'hoho'}}, false,  
true);
```

相当于: update users set age = age + 50, name = 'hoho' where name = 'Lisi';

3、删除

```
db.users.remove({age: 132});
```

4、查询修改删除

```
db.users.findAndModify({  
    query: {age: {$gte: 25}},  
    sort: {age: -1},  
    update: {$set: {name: 'a2'}, $inc: {age: 2}},  
    remove: true  
});
```

```
db.runCommand({ findandmodify : "users",  
    query: {age: {$gte: 25}},  
    sort: {age: -1},  
    update: {$set: {name: 'a2'}, $inc: {age: 2}},  
    remove: true  
});
```

原因是 新版的mongodb已经不支持addUser方法了。

改成createUser了。

```
db.createUser(  
    {  
        user: "accountUser",  
        pwd: "password",  
        roles: [ "readWrite", "dbAdmin" ]  
    }  
)
```

```
vi /etc/iptables.up.rules
#mongodb connect
-A INPUT -s 127.0.0.1 -p tcp --destination-port 27017 -m state --state
NEW,ESTABLISHED -j ACCEPT
-A OUTPUT -d 127.0.0.1 -p tcp --source-port 27017 -m state --state
ESTABLISHED -j ACCEPT
```

```
Sudo iptables-restore < /etc/iptables.up.rules
```

```
Sudo service mongod stop
Sudo service mongod start
Sudo service mongod restart
netstat -an|grep 27017
处理方法：终端中输入：
ps aux | grep mongod
找到对应进程id,杀死：
kill -9 pid
```

mongodb常用命令：

1、查询库、查询表

```
show dbs //查询所有的数据库
```

```
show collections //查询当前数据库下的所有数据表
```

2、建库和删库

```
use myDbs //建立一个名为myDbs的数据库，当这个库存在时则是切换到这个数据
库中去
```

```
use myDbs
```

```
db.dropDatabase(); //这两句是删除这个数据库
```

3、建表和删表

```
//表操作都是要先到一个数据库中去，通过use方法
db.myTable.insert({name:'hf',age:20}); //在mongodb中在插入数据时即创建了改
表，此时创建的是名为myTable的数据表
db.myTable.drop(); //删除myTable这个数据表
//如果没有指定数据库，表会创建在mongodb默认数据库test里
```

4、单表的增删改

```
db.myTable.insert({name:'hahaha',age:12}); //新增
```

```
db.myTable.update({name:'hf'},{$set:{age:25}}) //修改
```

```
db.myTable.remove({name:'hf'}); //删除
```

5、查询

```
db.myTable.find(); //查询myTable中的所有数据
```

```
db.myTable.find().sort({age:1}) //根据age升续
```

```
db.myTable.find().count(); //查询
```

```
for(var i=1;i<10;i++) db.myTable.save({name:"111",password:"123"})
i={name:"wpz1",password:"123"} db.myTable.save(i)
```

● 6、条件数据查询

条件数据查询，就是在查询中加入过滤条件，mongodb的精确过滤条件是制定查询数据中json数据。例如 db.user.find({"age": "20"}) 相当于sql中的 select

- * from user where age = '20'
- db.user.find({"age": "20"}) 这句话结合上面的普通查询，就可以查询出大部分想要的数据，但是有时候我们需要指定查询的字段：
-
- 当然还有另外一种方式做数组
还有一种直接打印json形式的查询方式

● 3, mongodb为了减少游标的内存开销还提供了findOne()方法，当然，方法内可以加过滤条件

```
. db.user.findOne()
. db.user.findOne({"name": "wpz"})
```

这两种写法都是合法的。

4, mongodb还提供limit来限制条数

```
. db.user.find().limit(2)
. db.user.find({"name": "wpz"}).limit(2)
```

- 5, 条件符查询 mongodb支持 < <= > >= 四种运算符查询

- db.user.find({"age": {\$gt:30}}) age大于30
- db.user.find({"age": {\$lt:30}}) age小于30
- db.user.find({"age": {\$gte:30}}) age大于或等于30
- db.user.find({"age": {\$lte:30}}) age小于或等于30

- 多条件查询

```
db.user.find({"age": {$gt:10,
    $lte:30}})
```

- 6, 匹配所有

\$all 这个操作符号类似于sql中的in运算符，但是不同的是in只需要满足一个值，但是all需要满足所有值。

```
db.user.find({"age": {$all:[6,8]});
```

- 7, 查询某一个字段是否存在：\$exists

```
db.user.find({"password": {$exists:true}}); password存在的记录
```

- db.user.find({"password":{\$exists:false}}); password不存在的记录
- 8,null值得处理
 - null处理比较奇怪，因为mongodb中的数据集合不能指定特定的格式，没有sql中的字段的概念，就是说，在同一个集合中有的字段在一条数据中存在，在另一条数据中不存在，所以，要找出改字段是不是为空，先要判断这个字段是不是存在才行。
 - db.user.find({age:{“in”:[null],“exists”:true}});
- 9, 取模运算 \$mod
 - 查询所有age取模10之后为0 的数据，即查询age为10的倍数的字段：
 - db.user.find({age:{\$mod:[10,0]});
- 10,不等于 \$ne -> (not equals)
 - 查询所有age不等于10 的数据
 - db.user.find({age:{\$ne:10}});
- 11,包含 \$in
 - 查询所有age等于10 或者20 的数据
 - db.user.find({age:{\$in:[10,20]});
- 12,不包含 \$nin
 - 查询所有age不等于10 或者20 的数据
 - db.user.find({age:{\$nin:[10,20]});
- 13,数组元素的个数 \$size
 - 查询age数据元素个数为3的数据
 - db.user.find({age:{\$size:3}});
- 14, 正则表达式匹配查询
 - name不以wpz开头的数据


```
db.user.find({"name":{$not:/^wpz.*/}});
```

15,count查询条数

- db.user.find().count();

16,skip 设置查询数据的起点

查询从第三条数据之后的五条数据

- db.user.find().skip(3).limit(5);

17 排序 sort

- db.user.find().sort({age:1}); 按照age升序

- db.user.find().sort({age:-1}); 按照age降序

mongodb也支持存储过程的查询。

1. 7,数据修改更新

2. mongodb的修改是比较烦的一种，要用到\$set:

例如，吧mongodb中,name为wpz,修改为 wpz_new

```
db.user.update({"name":"wpz"},{$set:"name":"wpz_new"});
```

1. 8, 数据删除

mongodb的删除比较简单，格式如下：

```
db.user.remove({"name":"wpz"});
```

● 1, mongodb启动服务：

命令行方式启动：

- . 直接cmd下进入mongodb的bin目录，输入mongod.exe –dbpath d:
 - \mongodb\db 启动成功
- 配置文件方式启动:
- .对于一些大神来说，普通的启动满足不了心里需求（玩笑），对于专业的DBA来说，会在启动mongodb数据库实例的时候加进去好多参数以便使得系统运行更高效稳定，但是如果吧参数都加载mongod的后面会使得参数混乱且每次都得手动的敲进去，不好管理，mongodb为此提供了配置文件启动的方式
 - .新建配置文件mongodb.cnf 编辑内容 dbpath=d:\mongodb\data\db保存在bin的同级目录下，打开cmd 进入bin下，输入mongod.exe -f d:
 - \mongodb\mongodb.cnf 启动成功
- daemon方式启动
- .以上的方式都有一个明显的缺陷，即如果控制台的界面不小心被关闭，则mongodb的服务也会随之停止，这带来了极大的不方便和不安全，这也促使有了daemon启动方式，这种方式需要加上–fork 即可，但是如果用到fork参数就必须使用 –logpath参数指定日志文件路径。
 - .data文件夹下新建log文件夹，并在文件夹下新建log1.log文件。进入bin目录，输入 mongod.exe –dbpath d:\mongodb\data\db –logpath d:
 - \mongodb\data\log\log1.log –fork 启动成功。
- mongod参数说明：
- .dbpath 数据库路径
 - .logpath 日志目录
 - .logappend 错误日志采用追加模式
 - .bind_ip 对外服务绑定的ip 一般置为空，即绑定本地多有可用ip上。但是如果有需要可以单独制定
 - .port 对外服务端口 ,web管理端口一般在这个基础上+1000
 - .fork 服务在后台以daemon方式运行
 - .journal 开启日志功能，通过保存日志来尖山单机故障的恢复时间。
 - .syncdelay 系统同步刷新磁盘的时间
 - .directoryperdb 每个db存放在单独的目录中，建议设置该参数
 - .maxConns 最大连接数
 - .repairpath 执行repair时的临时路径，在如果没有开启journal的时候，异常down机后，必须执行repair重启，否则会报错，上一篇文章有提到。
- 2, **mongodb停止服务 :**
 - Control-C
 - db.shutdownServer()
 - 值得注意的是，最好不要用linux的kill命令来杀死进程，这样会导致数据库数据的损坏。
 - 3, **新建数据库**

在mongodb中，并没有单独新建数据库的命令，mongodb是非关系型数据库中

 - 最像关系型数据库的数据库，所以，有很多命令也和sql有一定的相似之处。
 - 打开数据库连接：进入到bin目录 输入命令： mongo.exe
 - 查询所有的数据库 show dbs 或者 show databases

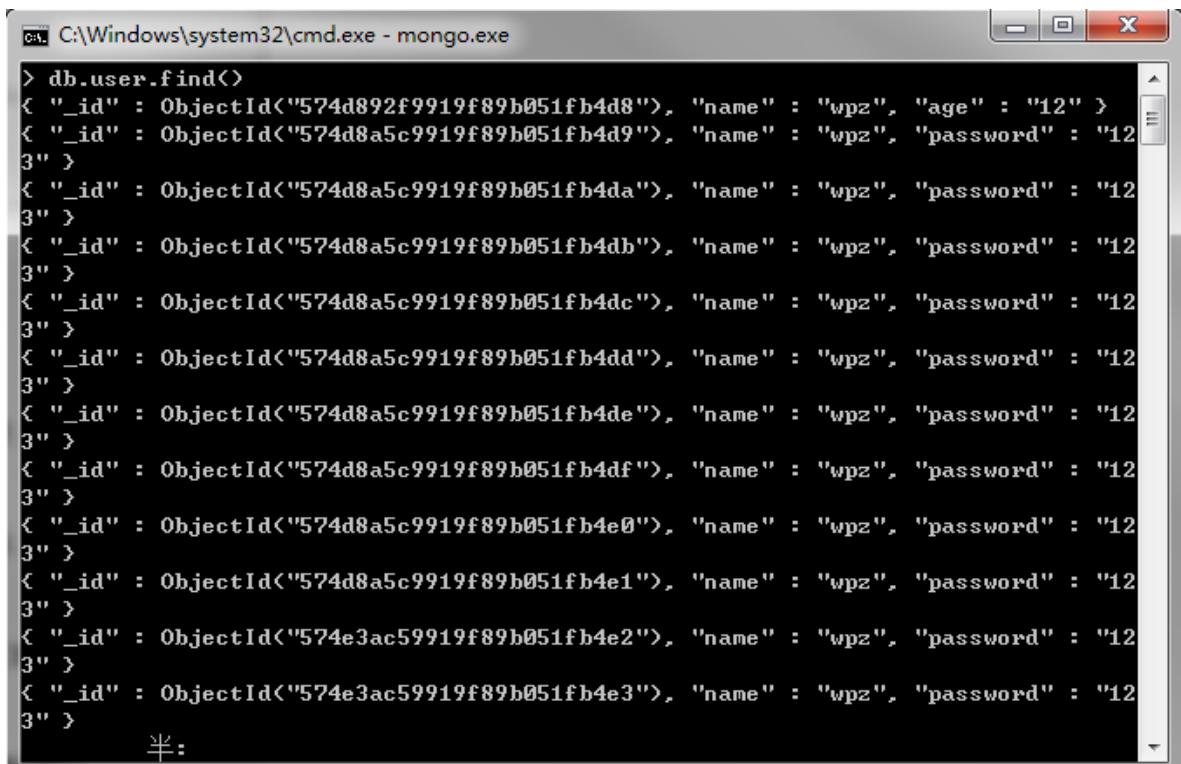
当我们想新建一个没有的数据库的时候，例如我们想新建数据库 testTwo ,直接可以输入命令 use testTwo即可，在testTwo存在的情况下回切换到testTwo，在不存在的情况下就会新建这个数据库。
 - 4, 插入记录

- mongodb之下和sql表对应的叫做集合， mongo中没有表的概念，只有集合的概念，和数据库类似，集合也无需单独的新建，在插入记录的同时，如果存在集合则插入数据到该集合中，如果不存在集合，则新建集合。

- db.user.save({“name”：“wpz”, “password”：“123”});
- 上面的命令代表新建user集合 并插入数据{“name”：“wpz”, “password”：“123”}
- 也可以定义为变量存储
- i={“name”：“wpz”, “password”：“123”}; db.user.save(i);
- 也可以用更灵活的for循环来存贮
- for(var i=1;i<10;i++) db.user.save({“name”：“wpz”, “password”：“123”});

● 5.普通数据查询

- mongodb的数据查询可以通过find()来查询，有几种常见的查询方式：
db.user.find() 查询出所有对象，这应该是最简单的全部查找的写法



```
> db.user.find()
[{"_id": ObjectId("574d892f9919f89b051fb4d8"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4d9"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4da"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4db"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4dc"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4dd"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4de"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4df"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4e0"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574d8a5c9919f89b051fb4e1"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574e3ac59919f89b051fb4e2"), "name": "wpz", "password": "123"}, {"_id": ObjectId("574e3ac59919f89b051fb4e3"), "name": "wpz", "password": "123"}]
半:
```

- 第二种， 使用游标来查询
- var cursor=db.user.find();
while(cursor.hasNext()) printjson(cursor.next());

```
C:\Windows\system32\cmd.exe - mongo.exe
3" >
> var cursor = db.user.find();
> while(cursor.hasNext())  printjson(cursor.next())
<
  "_id" : ObjectId("574d892f9919f89b051fb4d8"),
  "name" : "wpz",
  "age" : "12"
>
<
  "_id" : ObjectId("574d8a5c9919f89b051fb4d9"),
  "name" : "wpz",
  "password" : "123"
>
<
  "_id" : ObjectId("574d8a5c9919f89b051fb4da"),
  "name" : "wpz",
  "password" : "123"
>
<
  "_id" : ObjectId("574d8a5c9919f89b051fb4db"),
  "name" : "wpz",
  "password" : "123"
>
<
半:
```

第三种普通查询，游标可以看成为数组，可以查询数组上特定位置的值

```
C:\Windows\system32\cmd.exe - mongo.exe
> var cursor = db.user.find();
> while(cursor.hasNext())  printjson(cursor[4])
<
  "_id" : ObjectId("574d8a5c9919f89b051fb4dc"),
  "name" : "wpz",
  "password" : "123"
>
>
```

当然还有另外一种方式做数组

```
C:\Windows\system32\cmd.exe - mongo.exe
> var arr =db.user.find().toArray();
> printjson(arr[3]);
{
    "_id" : ObjectId("574d8a5c9919f89b051fb4db"),
    "name" : "wpz",
    "password" : "123"
}
>
>
```

还有一种直接打印json形式的查询方式

● 6. 条件数据查询

条件数据查询，就是在查询中加入过滤条件，mongodb的精确过滤条件是制定查询数据中json数据。例如 db.user.find({“age”:”20”}) 相当于sql中的 select

● * from user where age = ‘20’

db.user.find({“age”:”20”}) 这句话结合上面的普通查询，就可以查询出大部分想要的数据，但是有时候我们需要指定查询的字段：

```
C:\Windows\system32\cmd.exe - mongo.exe
> db.user.find({{"name":"wpz"}, {"password":true}}
< "_id" : ObjectId("574d892f9919f89b051fb4d8") ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4d9") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4da") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4db") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4dc") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4dd") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4de") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4df") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4e0") , "password" : "123" ,
< "_id" : ObjectId("574d8a5c9919f89b051fb4e1") , "password" : "123" ,
< "_id" : ObjectId("574e3ac59919f89b051fb4e2") , "password" : "123" ,
< "_id" : ObjectId("574e3ac59919f89b051fb4e3") , "password" : "123" ,
< "_id" : ObjectId("574e3ac59919f89b051fb4e4") , "password" : "123" ,
< "_id" : ObjectId("574e3ac59919f89b051fb4e5") , "password" : "123" ,
< "_id" : ObjectId("574e3ac59919f89b051fb4e6") , "password" : "123" ,
< "_id" : ObjectId("574e3ac59919f89b051fb4e7") , "password" : "123" ,
```

3, mongodb为了减少游标的内存开销还提供了findOne()方法，当然，方法内可以加过滤条件

. db.user.findOne()

. db.user.findOne({“name”:”wpz”})

这两种写法都是合法的。

4, mongodb还提供limit来限制条数

. db.user.find().limit(2)

. db.user.find({“name”:”wpz”}).limit(2)

○ 5, 条件符查询 mongodb支持< <= > >= 四种运算符查询

○ db.user.find({“age”:{\$gt:30}}) age大于30

○ db.user.find({“age”:{\$lt:30}}) age小于30

○ db.user.find({“age”:{\$gte:30}}) age大于或等于30

db.user.find({“age”:{\$lte:30}}) age小于或等于30

多条件查询

- ```
db.user.find({"age":{gt:10,
 ● lte:30}})
```
- 6, 匹配所有
 

\$all 这个操作符号类似于sql中的in运算符，但是不同的是in只需要满足一个值，但是all需要满足所有值。

```
db.user.find({"age":{$all:[6,8]});
```
- 7,查询某一个字段是否存在: \$exists
  - db.user.find({"password":{\$exists:true}}); password存在的记录
  - db.user.find({"password":{\$exists:false}}); password不存在的记录
- 8,null值得处理
 

null处理比较奇怪，因为mongodb中的数据集合不能指定特定的格式，没有sql中的字段的概念，就是说，在同一个集合中有的字段在一条数据中存在，在另一条数据中不存在，所以，要找出改字段是不是为空，先要判断这个字段是不是存在才行。

```
db.user.find({age:{“in”:[null],
 ● exists”:true}});
```
- 9, 取模运算 \$mod
 

查询所有age取模10之后为0 的数据，即查询age为10的倍数的字段：

  - db.user.find({age:{\$mod:[10,0]});
- 10,不等于 \$ne -> (not equals)
  - 查询所有age不等于10 的数据
  - db.user.find({age:{\$ne:10}});
- 11,包含 \$in
  - 查询所有age等于10 或者20 的数据
  - db.user.find({age:{\$in:[10,20]});
- 12,不包含 \$nin
  - 查询所有age不等于10 或者20 的数据
  - db.user.find({age:{\$nin:[10,20]});
- 13,数组元素的个数 \$size
  - 查询age数据元素个数为3的数据
  - db.user.find({age:{\$size:3}});
- 14, 正则表达式匹配查询
  - name不以wpz开头的数据

```
db.user.find({"name":{$not:/^wpz.*/}});
```

## 15,count查询条数

- db.user.find().count();

## 16,skip 设置查询数据的起点

查询从第三条数据之后的五条数据

- db.user.find().skip(3).limit(5);

## 17 排序 sort

- db.user.find().sort({age:1}); 按照age升序

- db.user.find().sort({age:-1}); 按照age降序

mongodb也支持存储过程的查询。

## 1. 7,数据修改更新

2. mongodb的修改是比较烦的一种，要用到\$set:

例如，吧mongodb中,name为wpz,修改为 wpz\_new

```
db.user.update({“name”：“wpz”}, {$set:“name”：“wpz_new”});
```

## 1. 8, 数据删除

mongodb的删除比较简单，格式如下：

```
db.user.remove({“name”：“wpz”});
```

## 1. 其他说明

在上面查询出的数据可以看出，所有的数据不管指定或者不指定都会带有\_id字段，这个字段相当于mongodb中的uuid，是数据唯一性的标示，可以理解为sql

### 2. 中的主键。

mongodb解压的目录：

| 名称               | 修改日期            | 类型     | 大小        |
|------------------|-----------------|--------|-----------|
| bsondump.exe     | 2013/8/18 20:26 | 应用程序   | 11,006 KB |
| mongo.exe        | 2013/8/18 18:47 | 应用程序   | 6,238 KB  |
| mongod.exe       | 2013/8/18 18:57 | 应用程序   | 11,061 KB |
| mongod.pdb       | 2013/8/18 18:57 | PDB 文件 | 89,779 KB |
| mongodump.exe    | 2013/8/18 19:13 | 应用程序   | 11,040 KB |
| mongoexport.exe  | 2013/8/18 19:32 | 应用程序   | 11,009 KB |
| mongofiles.exe   | 2013/8/18 20:17 | 应用程序   | 11,022 KB |
| mongoimport.exe  | 2013/8/18 19:41 | 应用程序   | 11,027 KB |
| mongooplog.exe   | 2013/8/18 20:08 | 应用程序   | 11,006 KB |
| mongoperf.exe    | 2013/8/18 20:36 | 应用程序   | 11,017 KB |
| mongorestore.exe | 2013/8/18 19:22 | 应用程序   | 11,032 KB |
| mongos.exe       | 2013/8/18 19:04 | 应用程序   | 8,644 KB  |
| mongos.pdb       | 2013/8/18 19:04 | PDB 文件 | 69,379 KB |
| mongostat.exe    | 2013/8/18 19:50 | 应用程序   | 11,037 KB |
| mongotop.exe     | 2013/8/18 19:59 | 应用程序   | 11,009 KB |

每一个exe都有对应的作用，相当于mongodb提供的一些常用的工具，例如，第一个是讲bson格式文件转存在json格式数据的工具，读者可自行查询每个工具的作用。

3, mongodb也提供了一些GUI管理工具，可视化的管理mongodb数据库。

.mongoVUE：提供mongodb的基本操作，优点是简单易用，但是总体上功能比较弱

.RockMongo 是windows平台首选的mongodb管理工具，速度快，安装简单，支持多种语言，支持多主机，可以保证数据的安全性。

.MongoHUB 是针对mac平台的mongodb管理工具。

## MongoDB Shell 常用操作命令

### ● 转载 2016年11月01日 11:34:31

- 标签：  
● **数据库 /**  
● **mongodb /**  
● **工作**  
1054

转载自：<http://blog.csdn.net/zhangzhebjut/article/details/>

# 13316761

MonoDB shell命令操作语法和JavaScript很类似，其实控制台底层的查询语句都是用JavaScript脚本完成操作的。

## Ø 数据库

1、Help查看命令提示

```
help
db.help();
db.yourColl.help();
db.youColl.find().help();
rs.help();
```

2、切换/创建数据库

```
>use yourDB;
```

当创建一个集合(table)的时候会自动创建当前数据库

3、查询所有数据库

```
show dbs;
```

4、删除当前使用数据库

```
db.dropDatabase();
```

5、从指定主机上克隆数据库

```
db.cloneDatabase("127.0.0.1");
```

将指定机器上的数据库的数据克隆到当前数据库

6、从指定的机器上复制指定数据库数据到某个数据库

```
db.copyDatabase("mydb", "temp", "127.0.0.1");
```

将本机的mydb的数据复制到temp数据库中

7、修复当前数据库

```
db.repairDatabase();
```

8、查看当前使用的数据库

```
db.getName();
```

```
db;
```

db和getName方法是一样的效果，都可以查询当前使用的数据库

9、显示当前db状态

```
db.stats();
```

10、当前db版本

```
db.version();
```

11、查看当前db的链接机器地址

```
db.getMongo();
```

## Ø Collection聚集集合

1、创建一个聚集集合 (table)

```
db.createCollection("collName", {size: 20, capped: 5, max: 100});
```

2、得到指定名称的聚集集合 (table)

```
db.getCollection("account");
```

3、得到当前db的所有聚集集合

```
db.getCollectionNames();
```

4、显示当前db所有聚集索引的状态

```
db.printCollectionStats();
```

## Ø 用户相关

1、添加一个用户

```
db.addUser("name");
```

```
db.addUser("userName", "pwd123", true);
```

添加用户、设置密码、是否只读

2、数据库认证、安全模式

```
db.auth("userName", "123123");
```

3、显示当前所有用户

```
show users;
```

4、删除用户

```
db.removeUser("userName");
```

## Ø 其他

1、查询之前的错误信息

```
db.getPrevError();
```

2、清除错误记录

```
db.resetError();
```

## 三、Collection聚集集合操作

### Ø 查看聚集集合基本信息

1、查看帮助

```
db.yourColl.help();
```

2、查询当前集合的数据条数

```
db.yourColl.count();
```

3、查看数据空间大小

```
db.userInfo.dataSize();
```

4、得到当前聚集集合所在的db

```
db.userInfo.getDB();
```

5、得到当前聚集的状态

```
db.userInfo.stats();
```

6、得到聚集集合总大小

```
db.userInfo.totalSize();
```

7、聚集集合储存空间大小

```
db.userInfo.storageSize();
```

8、Shard版本信息

```
db.userInfo.getShardVersion()
```

9、聚集集合重命名

```
db.userInfo.renameCollection("users");
```

将userInfo重命名为users

10、删除当前聚集集合

```
db.userInfo.drop();
```

## Ø 聚集集合查询

1、查询所有记录

```
db.userInfo.find();
```

相当于： select \* from userInfo;

默认每页显示20条记录，当显示不下的情况下，可以用it迭代命令查询下一页数据。

注意：键入it命令不能带“；”

但是你可以设置每页显示数据的大小，用DBQuery.shellBatchSize = 50;这样每页就显示50条记录了。

2、查询去掉后的当前聚集集合中的某列的重复数据

```
db.userInfo.distinct("name");
```

会过滤掉name中的相同数据

相当于： select distinct name from userInfo;

3、查询age = 22的记录

```
db.userInfo.find({"age": 22});
```

相当于： select \* from userInfo where age = 22;

4、查询age > 22的记录

```
db.userInfo.find({age: {$gt: 22}});
```

相当于： select \* from userInfo where age > 22;

5、查询age < 22的记录

```
db.userInfo.find({age: {$lt: 22}});
```

相当于： select \* from userInfo where age < 22;

6、查询age >= 25的记录

```
db.userInfo.find({age: {$gte: 25}});
```

相当于： select \* from userInfo where age >= 25;

7、查询age <= 25的记录

```
db.userInfo.find({age: {$lte: 25}});
```

8、查询age >= 23 并且 age <= 26

```
db.userInfo.find({age: {$gte: 23, $lte: 26}});
```

9、查询name中包含 mongo的数据

```
db.userInfo.find({name: /mongo/});
```

//相当于%%

```
select * from userInfo where name like '%mongo%';
```

10、查询name中以mongo开头的

```
db.userInfo.find({name: /^mongo/});
```

```
select * from userInfo where name like 'mongo%';
```

11、查询指定列name、 age数据

```
db.userInfo.find({}, {name: 1, age: 1});
```

相当于： select name, age from userInfo;

当然name也可以用true或false,当用ture的情况下和name:1效果一样，如果用false就是排除name，显示name以外的列信息。

12、查询指定列name、 age数据, age > 25

```
db.userInfo.find({age: {$gt: 25}}, {name: 1, age: 1});
```

相当于： select name, age from userInfo where age > 25;

13、按照年龄排序

升序： db.userInfo.find().sort({age: 1});

降序： db.userInfo.find().sort({age: -1});

14、查询name = zhangsan, age = 22的数据

```
db.userInfo.find({name: 'zhangsan', age: 22});
```

相当于： select \* from userInfo where name = 'zhangsan' and age = '22';

15、查询前5条数据

```
db.userInfo.find().limit(5);
```

相当于： select top 5 \* from userInfo;

16、查询10条以后的数据

```
db.userInfo.find().skip(10);
```

相当于： select \* from userInfo where id not in (

```
select top 10 * from userInfo
```

);

17、查询在5-10之间的数据

```
db.userInfo.find().limit(10).skip(5);
```

可用于分页， limit是pageSize, skip是第几页\*pageSize

18、or与查询

```
db.userInfo.find({$or: [{age: 22}, {age: 25}]});
```

相当于： select \* from userInfo where age = 22 or age = 25;

19、查询第一条数据

```
db.userInfo.findOne();
```

相当于： select top 1 \* from userInfo;

```
db.userInfo.find().limit(1);
```

20、查询某个结果集的记录条数

```
db.userInfo.find({age: {$gte: 25}}).count();
```

相当于： select count(\*) from userInfo where age >= 20;

21、按照某列进行排序

```
db.userInfo.find({$sex: {$exists: true}}).count();
```

相当于： select count(sex) from userInfo;

## Ø 索引

1、创建索引

```
db.userInfo.ensureIndex({name: 1});
```

```
db.userInfo.ensureIndex({name: 1, ts: -1});
```

2、查询当前聚集集合所有索引

```
db.userInfo.getIndexes();
```

3、查看总索引记录大小

```
db.userInfo.totalIndexSize();
```

4、读取当前集合的所有index信息

```
db.users.reIndex();
```

5、删除指定索引

```
db.users.dropIndex("name_1");
```

6、删除所有索引索引

```
db.users.dropIndexes();
```

## Ø 修改、添加、删除集合数据

1、添加

```
db.users.save({name: 'zhangsan', age: 25, sex: true});
```

添加的数据的数据列，没有固定，根据添加的数据为准

## 2、修改

```
db.users.update({age: 25}, {$set: {name: 'changeName'}}, false, true);
```

相当于： update users set name = ‘changeName’ where age = 25;

```
db.users.update({name: 'Lisi'}, {$inc: {age: 50}}, false, true);
```

相当于： update users set age = age + 50 where name = ‘Lisi’;

```
db.users.update({name: 'Lisi'}, {$inc: {age: 50}}, {$set: {name: 'hoho'}}, false, true);
```

相当于： update users set age = age + 50, name = ‘hoho’ where name = ‘Lisi’;

## 3、删除

```
db.users.remove({age: 132});
```

## 4、查询修改删除

```
db.users.findAndModify({
 query: {age: {$gte: 25}},
 sort: {age: -1},
 update: {$set: {name: 'a2'}, $inc: {age: 2}},
 remove: true
});
```

```
db.runCommand({ findandmodify : "users",
 query: {age: {$gte: 25}},
 sort: {age: -1},
 update: {$set: {name: 'a2'}, $inc: {age: 2}},
 remove: true
});
```

**update** 或 **remove** 其中一个必须是参数; 其他参数可选。

| 参数            | 详解                                            | 默认值        |
|---------------|-----------------------------------------------|------------|
| <b>query</b>  | 查询过滤条件                                        | {}         |
| <b>sort</b>   | 如果多个文档符合查询过滤条件，将以该参数指定的排列方式选择出排在首位的对象，该对象将被操作 | {}         |
| <b>remove</b> | 若为true，被选中对象将在返回前被删除                          | N/A        |
| <b>update</b> | 一个修改器对象                                       | N/A        |
| <b>new</b>    | 若为true，将返回修改后的对象而不是原始对象。在删除操作中，该参数被忽略。        | false      |
| <b>fields</b> | 参见Retrieving a Subset of Fields (1.5.0+)      | All fields |
| <b>upsert</b> | 创建新对象若查询结果为空。示例 (1.5.4+)                      | false      |

## Ø 语句块操作

### 1、简单Hello World

```
print("Hello World!");
```

这种写法调用了print函数，和直接写入"Hello World!"的效果是一样的；

## 2、将一个对象转换成json

```
tojson(new Object());
tojson(new Object('a'));
```

## 3、循环添加数据

```
> for (var i = 0; i < 30; i++) {
... db.users.save({name: "u_" + i, age: 22 + i, sex: i % 2});
...};
```

这样就循环添加了30条数据，同样也可以省略括号的写法

```
> for (var i = 0; i < 30; i++) db.users.save({name: "u_" + i, age: 22 + i, sex: i % 2});
```

也是可以的，当你用db.users.find()查询的时候，显示多条数据而无法一页显示的情况下，可以用it查看下一页的信息；

## 4、find 游标查询

```
> var cursor = db.users.find();
> while (cursor.hasNext()) {
 printjson(cursor.next());
}
```

这样就查询所有的users信息，同样可以这样写

```
var cursor = db.users.find();
while (cursor.hasNext()) { printjson(cursor.next()); }
```

同样可以省略{}号

## 5、forEach迭代循环

```
db.users.find().forEach(printjson);
```

forEach中必须传递一个函数来处理每条迭代的数据信息

## 6、将find游标当数组处理

```
var cursor = db.users.find();
cursor[4];
```

取得下标索引为4的那条数据

既然可以当做数组处理，那么就可以获得它的长度：cursor.length();或者cursor.count();

那样我们也可以用循环显示数据

```
for (var i = 0, len = c.length(); i < len; i++) printjson(c[i]);
```

## 7、将find游标转换成数组

```
> var arr = db.users.find().toArray();
> printjson(arr[2]);
```

用toArray方法将其转换为数组

## 8、定制我们自己的查询结果

```
只显示age <= 28的并且只显示age这列数据
db.users.find({age: {$lte: 28}}, {age: 1}).forEach(printjson);
db.users.find({age: {$lte: 28}}, {age: true}).forEach(printjson);
排除age的列
db.users.find({age: {$lte: 28}}, {age: false}).forEach(printjson);
```

## 9、forEach传递函数显示信息

```
db.things.find({x:4}).forEach(function(x) {print(tojson(x));});
```

上面介绍过forEach需要传递一个函数，函数会接受一个参数，就是当前循环的对象，然后在函数体中处理传入的参数

## Sequelize 中文文档 v4

### 命令行界面

#### 安装命令行界面

让我们从安装CLI开始，你可以在 [这里](#) 找到说明。最推荐的方式是这样安装

```
$ npm install --save sequelize-cli
```

#### 引导

要创建一个空项目，你需要执行 init 命令

```
$ node_modules/.bin/sequelize init
```

- 这将创建以下文件夹

- config, 包含配置文件，它告诉CLI如何连接数据库
- models, 包含您的项目的所有模型
- migrations, 包含所有迁移文件
- seeders, 包含所有种子文件

#### 结构

在继续进行之前，我们需要告诉 CLI 如何连接到数据库。为此，可以打开默认配置文件 config/config.json。看起来像这样

```
{
 development: {
 username: 'root',
 password: null,
 database: 'database_development',
 host: '127.0.0.1',
 dialect: 'mysql'
 },
 test: {
 username: 'root',
 password: null,
 database: 'database_test',
 host: '127.0.0.1',
 dialect: 'mysql'
 },
 production: {
 username: 'root',
```

```
password: null,
database: 'database_production',
host: '127.0.0.1',
dialect: 'mysql'
}
}
```

现在编辑此文件并设置正确的数据库凭据和方言。

**注意:** 如果你的数据库还不存在, 你可以调用 `db:create` 命令。 通过正确的访问, 它将为您创建该数据库。

## 创建第一个模型 (和迁移)

一旦您正确配置了CLI配置文件, 您就可以首先创建迁移。 它像执行一个简单的命令一样简单。

- 我们将使用 `model:generate` 命令。 此命令需要两个选项

- `name`, 模型的名称  
`attributes`, 模型的属性列表

让我们创建一个名叫 `User` 的模型

```
$ node_modules/.bin/sequelize model:generate --name User --attributes
firstName:string,lastName:string,email:string
```

- 这将发生以下事情

- 在 `models` 文件夹中创建了一个 `user` 模型文件  
在 `migrations` 文件夹中创建了一个名字像 `XXXXXXXXXXXXXX-create-user.js` 的迁移文件

**注意:** `Sequelize` 将只使用模型文件, 它是表描述。 另一边, 迁移文件是该模型的更改, 或更具体的是说 `CLI` 所使用的表。 处理迁移, 如提交或日志, 以进行数据库的某些更改。

## 运行迁移

直到这一步, `CLI` 没有将任何东西插入数据库。 我们刚刚为我们的第一个模型 `User` 创建了必需的模型和迁移文件。 现在要在数据库中实际创建该表, 需要运行 `db:migrate` 命令。

```
$ node_modules/.bin/sequelize db:migrate
```

- 此命令将执行这些步骤

- 将在数据库中确保一个名为 `SequelizeMeta` 的表。 此表用于记录在当前数据库上运行的迁移  
开始寻找尚未运行的任何迁移文件。 这可以通过检查 `SequelizeMeta` 表。 在这个例子中, 它将运行我们在最后一步中创建的 `XXXXXXXXXXXXXX-create-user.js` 迁移。  
创建一个名为 `User` 的表, 其中包含其迁移文件中指定的所有列。

## 撤消迁移

现在我们的表已创建并保存在数据库中。 通过迁移, 只需运行命令即可恢复为旧状态。

您可以使用 `db:migrate:undo`, 这个命令将会恢复最近的迁移。

```
$ node_modules/.bin/sequelize db:migrate:undo
```

通过使用 `db:migrate:undo:all` 命令撤消所有迁移, 可以恢复到初始状态。 您还可以通过将其名称传递到 `--to` 选项中来恢复到特定的迁移。

```
$ node_modules/.bin/sequelize db:migrate:undo:all --to XXXXXXXXXXXXXXXX-create-posts.js
```

## 创建第一个种子

假设我们希望在默认情况下将一些数据插入到几个表中。如果我们跟进前面的例子，我们可以考虑为 User 表创建演示用户。

要管理所有数据迁移，您可以使用 seeders。种子文件是数据的一些变化，可用于使用样本数据或测试数据填充数据库表。

让我们创建一个种子文件，它会将一个演示用户添加到我们的 User 表中。

```
$ node_modules/.bin/sequelize seed:generate --name demo-user
```

这个命令将会在 seeders 文件夹中创建一个种子文件。文件名看起来像是 XXXXXXXXXXXXX-demo-user.js，它遵循相同的 up/down 语义，如迁移文件。

现在我们应该编辑这个文件，将演示用户插入User表。

```
'use strict';
```

```
module.exports = {
 up: (queryInterface, Sequelize) => {
 return queryInterface.bulkInsert('Users', [
 {
 firstName: 'John',
 lastName: 'Doe',
 email: 'demo@demo.com'
 }
], {});
 },
 down: (queryInterface, Sequelize) => {
 return queryInterface.bulkDelete('Users', null, {});
 }
};
```

## 运行种子

在上一步中，你创建了一个种子文件。但它还没有保存到数据库。为此，我们需要运行一个简单的命令。

```
$ node_modules/.bin/sequelize db:seed:all
```

这将执行该种子文件，您将有一个演示用户插入 User 表。

注意: seeders 执行不会存储在任何使用 SequelizeMeta 表的迁移的地方。如果你想覆盖这个，请阅读 存储 部分

## 撤销种子

Seeders 如果使用了任何存储那么就可以被撤消。有两个可用的命令

如果你想撤消最近的种子

```
node_modules/.bin/sequelize db:seed:undo
```

如果你想撤消所有的种子

```
node_modules/.bin/sequelize db:seed:undo:all
```

## 高级专题

## 迁移框架

以下框架显示了一个典型的迁移文件。

```
module.exports = {
 up: (queryInterface, Sequelize) => {
 // 转变为新状态的逻辑
 },
}
```

```

down: (queryInterface, Sequelize) => {
 // 恢复更改的逻辑
}
}

传递的 queryInterface 对象可以用来修改数据库。 Sequelize 对象存储可用的数据类型，如 STRING 或 INTEGER。 函数 up 或 down 应该返回一个 Promise 。 让我们来看一个例子

module.exports = {
 up: (queryInterface, Sequelize) => {
 return queryInterface.createTable('Person', {
 name: Sequelize.STRING,
 isBetaMember: {
 type: Sequelize.BOOLEAN,
 defaultValue: false,
 allowNull: false
 }
 });
 },
 down: (queryInterface, Sequelize) => {
 return queryInterface.dropTable('Person');
 }
}

```

### .sequelizerc 文件

这是一个特殊的配置文件。 它允许您指定通常作为参数传递给CLI的各种选项。 在

- 某些情况下，您可以使用它。

- 你想要覆盖到 migrations, models, seeders 或 config 文件夹的路径.

你想要重命名 config.json 成为别的名字比如 database.json

还有更多的，让我们看一下如何使用这个文件进行自定义配置。

对于初学者，可以在项目的根目录中创建一个空文件。

\$ touch .sequelizerc

现在可以使用示例配置。

```
const path = require('path');
```

```

module.exports = {
 'config': path.resolve('config', 'database.json'),
 'models-path': path.resolve('db', 'models'),
 'seeders-path': path.resolve('db', 'seeders'),
 'migrations-path': path.resolve('db', 'migrations')
}

```

- 通过这个配置你告诉CLI:

- 使用 config/database.json 文件来配置设置
- 使用 db/models 作为模型文件夹
- 使用 db/seeders 作为种子文件夹
- 使用 db/migrations 作为迁移文件夹

## 动态配置

配置文件是默认的一个名为 config.json 的JSON文件。但有时你想执行一些代码或访问环境变量，这在JSON文件中是不可能的。

Sequelize CLI可以从“JSON”和“JS”文件中读取。这可以用.sequelizerc文件设置。让我们来看一下

首先，您需要在项目的根文件夹中创建一个 .sequelizerc 文件。该文件应该覆盖 JS 文件的配置路径。推荐这个

```
const path = require('path');
```

```
module.exports = {
 'config': path.resolve('config', 'config.js')
}
```

现在，Sequelize CLI将加载 config/config.js 以获取配置选项。由于这是一个JS文件，您可以执行任何代码并导出最终的动态配置文件。

一个 config/config.js 文件的例子

```
const fs = require('fs');
```

```
module.exports = {
 development: {
 username: 'database_dev',
 password: 'database_dev',
 database: 'database_dev',
 host: '127.0.0.1',
 dialect: 'mysql'
 },
 test: {
 username: 'database_test',
 password: null,
 database: 'database_test',
 host: '127.0.0.1',
 dialect: 'mysql'
 },
 production: {
 username: process.env.DB_USERNAME,
 password: process.env.DB_PASSWORD,
 database: process.env.DB_NAME,
 host: process.env.DB_HOSTNAME,
 dialect: 'mysql',
 dialectOptions: {
 ssl: {
 ca: fs.readFileSync(__dirname + '/mysql-ca-master.crt')
 }
 }
 }
};
```

## 使用环境变量

使用CLI，您可以直接访问 config/config.js 内的环境变量。您可以使

用 `.sequelizerc` 来告诉 CLI 使用 `config/config.js` 进行配置。这在上一节中有所解释。

然后你可以使用正确的环境变量来暴露文件。

```
module.exports = {
 development: {
 username: 'database_dev',
 password: 'database_dev',
 database: 'database_dev',
 host: '127.0.0.1',
 dialect: 'mysql'
 },
 test: {
 username: process.env.CI_DB_USERNAME,
 password: process.env.CI_DB_PASSWORD,
 database: process.env.CI_DB_NAME,
 host: '127.0.0.1',
 dialect: 'mysql'
 },
 production: {
 username: process.env.PROD_DB_USERNAME,
 password: process.env.PROD_DB_PASSWORD,
 database: process.env.PROD_DB_NAME,
 host: process.env.PROD_DB_HOSTNAME,
 dialect: 'mysql'
 }
}
```

## 指定方言选项

有时你想指定一个 `dialectOption`, 如果它是一个通用配置, 你可以将其添加到 `config/config.json` 中。有时你想执行一些代码来获取 `dialectOptions`, 你应该为这些情况使用动态配置文件。

```
{
 "production": {
 "dialect": "mysql",
 "dialectOptions": {
 "bigNumberStrings": true
 }
 }
}
```

## 生产用途

有关在生产环境中使用 CLI 和迁移设置的一些提示。

1) 使用环境变量进行配置设置。这是通过动态配置更好地实现的。样品生产安全配置可能看起来像

```
const fs = require('fs');
```

```
module.exports = {
 development: {
 username: 'database_dev',
```

```

 password: 'database_dev',
 database: 'database_dev',
 host: '127.0.0.1',
 dialect: 'mysql'
},
test: {
 username: 'database_test',
 password: null,
 database: 'database_test',
 host: '127.0.0.1',
 dialect: 'mysql'
},
production: {
 username: process.env.DB_USERNAME,
 password: process.env.DB_PASSWORD,
 database: process.env.DB_NAME,
 host: process.env.DB_HOSTNAME,
 dialect: 'mysql',
 dialectOptions: {
 ssl: {
 ca: fs.readFileSync(__dirname + '/mysql-ca-master.crt')
 }
 }
}
};

```

我们的目标是为各种数据库秘密使用环境变量，而不是意外检查它们来源控制。

## 存储

- 可以使用三种类型的存储：sequelize, json和none。
  - sequelize : 将迁移和种子存储在 sequelize 数据库的表中
  - json : 将迁移和种子存储在json文件上
  - none : 不存储任何迁移/种子

## 迁移存储

默认情况下，CLI 将在您的数据库中创建一个名为 SequelizeMeta 的表，其中包含每个执行迁移的条目。要更改此行为，可以在配置文件中添加三个选项。使用 migrationStorage 可以选择要用于迁移的存储类型。如果选择 json，可以使用 migrationStoragePath 指定文件的路径，或者 CLI 将写入 sequelize-meta.json 文件。如果要将数据保存在数据库中，请使用 sequelize，但是要使用其他表格，可以使用 migrationStorageTableName.

```
{
 "development": {
 "username": "root",
 "password": null,
 "database": "database_development",
 "host": "127.0.0.1",
 "dialect": "mysql",
 }
}
```

```
// 使用不同的存储类型. Default: sequelize
"migrationStorage": "json",

// 使用不同的文件名. Default: sequelize-meta.json
"migrationStoragePath": "sequelizeMeta.json",

// 使用不同的表名. Default: SequelizeMeta
"migrationStorageTableName": "sequelize_meta"
}

}

注意: 不推荐使用 none 存储作为迁移存储。如果您决定使用它, 请注意将会没有任何移动记录或没有运行的记录。
```

## 种子储存

默认情况下, CLI 不会保存任何被执行的种子。如果您选择更改此行为(!), 则可以在配置文件中使用 `seederStorage` 来更改存储类型。如果选择 `json`, 可以使用 `seederStoragePath` 指定文件的路径, 或者 CLI 将写入文件 `sequelize-data.json`。如果要将数据保存在数据库中, 请使用 `sequelize`, 您可以使用 `seederStorageTableName` 指定表名, 否则将默认为 `SequelizeData`。

```
{
 "development": {
 "username": "root",
 "password": null,
 "database": "database_development",
 "host": "127.0.0.1",
 "dialect": "mysql",
 // 使用不同的存储空间. Default: none
 "seederStorage": "json",
 // 使用不同的文件名. Default: sequelize-data.json
 "seederStoragePath": "sequelizeData.json",
 // 使用不同的表名 Default: SequelizeData
 "seederStorageTableName": "sequelize_data"
 }
}
```

## 配置连接字符串

作为 `--config` 选项的替代方法, 可以使用定义数据库的配置文件, 您可以使用 `--url` 选项传递连接字符串。例如:

```
$ node_modules/.bin/sequelize db:migrate --url 'mysql://
root:password@mysql_host.com/database_name'
```

## 通过SSL连接

确保 `ssl` 在 `dialectOptions` 和基本配置中指定。

```
{
 "production": {
 "dialect": "postgres",
 "ssl": true,
 "dialectOptions": {
 "ssl": true
 }
 }
}
```

```
 }
 }
}
```

## 程序化使用

Sequelize 有一个 [姊妹库](#)，用于以编程方式处理迁移任务的执行和记录。

## 查询界面

使用 queryInterface 对象描述之前，您可以更改数据库模式。 查看完整的公共方法列表，它支持 [QueryInterface API](#)

## 下载PHP文件（随便）

<http://php.net/downloads.php>

选择对应的版本，php7.1.x，下载tar.bz

在Application/MAMP/bin/php/php7.1.6/下新建一个include文件夹

然后把刚刚的下载的包，放进去，然后解压，改名为php

然后在终端cd到php文件夹，输入

./configure

Phppredis 出错时

解决**configure: WARNING: You will need re2c 0.13.4 or later**

解决方法.

```
1 下载https://nchc.dl.sourceforge.net/project/re2c/1.0.1/re2c-1.0.1.tar.gz
2 tar zxvf re2c-1.0.1.tar.gz && cd re2c-1.0.1
3 ./configure
4 make && make install
```

Cd /Applications/MAMP/bin/php/php7.1.6/include/php

1 /Applications/MAMP/bin/php/php7.1.6/include/php/ext/zlib

2 cp config0.m4 ../../config.m4

3 /Applications/MAMP/bin/php/php7.1.6/bin/phpize --with-php-config=/
Applications/MAMP/bin/php/php7.1.6/bin/php-config

4 ./configure

```
sudo cp -p modules/redis.so /Applications/MAMP/bin/php/php7.0.0/lib/php/
extensions/no-debug-non-zts-20151012/
```

## 下载Redis拓展

在GitHub的phppredis仓库下载PHP7.0的包。

git clone <https://github.com/nicolasff/phppredis>

brew install php56-redis

```
brew install php71-redis
```

## laravel 如何使用composer自动加载自己定义的文件夹

### 一. 问题

当我们 clone下来一个laravel框架，接着就开始我们表演，但是我们根据业务需求需要创建一些自定义的文件夹，那么我们该如何加载他们呢，如何避免这类错误

```
[Symfony\Component\Debug\Exception\FatalThrowableError] Class 'tools\alyduanxin\api_demo\SmsDemo' not found
```

### ● 二：分三步来解决这个问题

在laravel 中项目根目录下创建自己的文件夹,例如我在项目根目录下创建了一个

- tools文件夹。

在项目文件夹的根目录下找到composer.json文件，在autoload里添加psr-4节点

- "psr-4": {  
 "tools\\": "tools/"  
}

接着执行

```
composer dump-autoload -o
```

- 

至此我们就加载成功啦

### 1.问题描述

在Laravel中引入了一个第三方验证码类Code.class.php，在使用的时候发现如果不给这个类设置命名空间，那么需要在使用时用require引入这个文件，引入后在

new Code()时会报Class 'App\Http\Controllers\Admin\Code' not found，即找不到这个类的错误，发现系统在当前文件的命名空间去找这个类。所以需要在new时类名前加\，即new \Code()，这样会在根命名空间下找Code类就可以正常使用。

好奇心下给Code类添加了个命名空间namespace resources\org\code;，发现在使用时use resources\org\code\Code;，依然报错找不到这个类。

### 2.解决过程

通过修改命名空间，改变类文件存放位置，都不能解决这个错误，百度后发现应该自定义类的加载问题，即命名空间使用正确，但是laravel没有加载这个类文件。最后在csdn大神博客发现解决办法。

在composer.json的autoload内的classmap项新增类包，

```
"autoload": {
 "classmap": [
 "database",
 "resources/org/code"//添加自己的命名空间
],
 "psr-4": {
 "App\\": "app/"
 }
},
```

完成之后还需要在命令行使用composer命令

```
composer dump-autoload
```

做完这些再回到项目中，在使用Code类的文件中use resources\org\code\Code;就可以正确的new Code()了。

### 3.总结

在自定义类不使用命名空间时可以直接require类文件进行使用，但在new \Code时要加上\指明根命名空间。

当使用命名空间时要进行配置，告诉系统进行自动加载自定义类包，然后可以使用use。

另外经过测试发现当使用命名空间时仅使用require仍然报错，还要指明命名空间才可以。但使用了use可以省去require。

CAS： CAS(Central Authentication Service)是一款不错的针对 Web 应用的单点登录框架，这里介绍下我刚在laravel5上搭建成功的cas。提前准备工作：可运行的laravel5的工程，cas的服务器端已经存在。

环境：[linux](#) (Ubuntu)

一， 下[◆phpcas](#)源代码。

在laravel5的项目app目录下创建library目录，下载phpcas库，git clone https://github.com/Jasig/phpCAS.git，clone下来是一个phpcas的文件目录。

## 二， 创建provider

在app下创建目录cas，创建CasAuthProvider.php，内容如下：

```
<?php
namespace cas;
use Illuminate\Contracts\Auth\UserProvider;
use Illuminate\Contracts\Hashing\Hasher;
use Illuminate\Contracts\Auth\Authenticatable;
use Illuminate\Auth\GenericUser;
class CasAuthProvider implements UserProvider {
/**
 * Retrieve a user by their unique identifier.
 *
 * @param mixed $id
 * @return \Illuminate\Auth\UserInterface|null
 */
public function retrieveById($id) {
 return $this->caSUSEr();
}
/**
 * Retrieve a user by the given credentials.
 *
 * @param array $credentials
 * @return \Illuminate\Auth\UserInterface|null
 */
public function retrieveByCredentials(array $credentials) {
 return $this->casUser();
}
/**
 * Validate a user against the given credentials.
 *
 * @param \Illuminate\Auth\UserInterface $user
 * @param array $credentials
 * @return bool
 */
public function validateCredentials(Authenticatable $user, array $credentials) {
 return true;
}
protected function casUser() {
 $cas_host = \Config::get('app.cas_host');
 //dump($cas_host);
 $cas_context = \Config::get('app.cas_context');
 $cas_port = \Config::get('app.cas_port');
 \phpCAS::setDebug();
 \phpCAS::client(CAS_VERSION_2_0, $cas_host, $cas_port, $cas_context);
```

```
\phpCAS::setNoCasServerValidation();
if (\phpCAS::isAuthenticated()) {
 $attributes = array(
 'id' => \phpCAS::getUser(),
 'name' => \phpCAS::getUser()
);
 return new GenericUser($attributes);
} else {
 //\phpCAS::setServerURL(\Config::get('app.url'));
 \phpCAS::forceAuthentication();
}
return null;
}
/**
 * Needed by Laravel 4.1.26 and above
 */
public function retrieveByToken($identifier, $token) {
 return new \Exception('not implemented');
}
/**
 * Needed by Laravel 4.1.26 and above
 */
public function updateRememberToken(Authenticatable $user, $token) {
 return new \Exception('not implemented');
}
}
}
?>
```

### 三，修改config

在config/app.php中添加如下三个配置项：

```
'cas_host'=>'*****', //认证服务器
'cas_context'=>",//还没弄明白是什么
'cas_port'=>000,//认证服务端口
'url'=>'http://localhost/>,
```

### 四，加载认证库

在app/providers/AppServiceProvider.php里，在类AppServiceProvider的register函数里添加认证方式：

```
Auth::extend('cas', function($app) {
 return new CasAuthProvider;
});
```

修改app/config/auth.php认证driver: 'driver' => 'cas',

在composer.json里配置加载项，在autoload里的classmap中添加如下路径：

```
"autoload": {
 "classmap": [

]
}
```

```
"app/library",
"app/library/phpCAS",
"app/cas"
```

```
]
```

```
}
```

在项目根目录下执行： composer dump-autoload

五， 实现

在app/http/controllers/下创建CasAuthController.php， 添加login和logout方法：

```
public function login() {
```

```
 $message_error = "";
```

```
 if (Auth::check()) {
```

```
 } else {
```

```
 if (Auth::attempt(array())) {
```

```
 // Redirect to link after login
```

```
 }
```

```
 // Redirect to un-logged in page
```

```
 }
```

```
 dump(\phpCAS::getUser());
```

```
 dump(Auth::user());
```

```
}
```

```
 public function logout() {
```

```
 $cas_host = \Config::get('app.cas_host');
```

```
 //dump($cas_host);
```

```
 $cas_context = \Config::get('app.cas_context');
```

```
 $cas_port = \Config::get('app.cas_port');
```

```
 \phpCAS::setDebug();
```

```
 \phpCAS::client(CAS_VERSION_2_0, $cas_host, $cas_port, $cas_context);
```

```
 \phpCAS::setNoCasServerValidation();
```

```
 \phpCAS::logoutWithRedirectService(\Config::get('app.url'));
```

```
}
```

在routes.php里添加路由规则就OK了， 把项目默认的登陆和注销方法指到这里来，  
当login的时候， 会出现服务器的登陆页面。

有个问题， 就是这么改动之后， 原来我设置的不需要登陆就能浏览的页面， 现在进入的时候也会跳出登陆页面， 不知道为什么， 希望高手指导下， 谢谢！

User.php

```
<?php
```

```
namespace App;
```

```
use Illuminate\Notifications\Notifiable;
```

```
use Illuminate\Foundation\Auth\User as Authenticatable;
```

```
class User extends Authenticatable
```

```
{
```

```
 use Notifiable;
```

```
 /**
```

```

 * The attributes that are mass assignable.
 *
 * @var array
 */
protected $fillable = [
 'name', 'email', 'password',
];

/**
 * The attributes that should be hidden for arrays.
 *
 * @var array
 */
protected $hidden = [
 'password', 'remember_token',
];
}

```

LoginController.php

```

<?php

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;
use Illuminate\Foundation\Auth\AuthenticatesUsers;

class LoginController extends Controller
{
 /*
 |--
 | Login Controller
 |--
 |
 | This controller handles authenticating users for the application and
 | redirecting them to your home screen. The controller uses a trait
 | to conveniently provide its functionality to your applications.
 |
 */
 use AuthenticatesUsers;

 /**
 * Where to redirect users after login.
 *
 * @var string
 */

```

```

*/
protected $redirectTo = '/home';

/**
 * Create a new controller instance.
 *
 * @return void
 */
public function __construct()
{
 $this->middleware('guest')->except('logout');
}
}

```

## 向军 laravel宝典

### 安装增加代码提示工具

- 1 composer require barryvdh/laravel-ide-helper
- 2 增加服务提供者 文件在 app/config/app.php 中 , 增加下面的一行  
Barryvdh\LaravelIdeHelper\IdeHelperServiceProvider::class,
- 3 php artisan ide-helper:generate
- 4 重启工具才会起作用

解决mysql 版本过低(mysql5.7以下 php artisan migrate 出错如下)

SQLSTATE[42000]: Syntax error or access violation: 1071 Specified key was too long;

1种解决方式是更改 config/database.php mysql数据库的设置

```
'charset' => 'utf8mb4', //改成utf8
'collation' => 'utf8mb4_unicode_ci', //改成utf8_unicode_ci'
```

2种解决方式是在 app/Providers/AppServiceProvider.php 中boot中增加 Schema::defaultStringLength(191);代码

如下:

```
<?php
```

```
namespace App\Providers;
```

```
use Illuminate\Support\ServiceProvider;
use Schema;
```

```
class AppServiceProvider extends ServiceProvider
{
 /**
 * Bootstrap any application services.
 *

```

```

 * @return void
 */
public function boot()
{
 //
 Schema::defaultStringLength(191);
}

/**
 * Register any application services.
 *
 * @return void
 */
public function register()
{
 //
}
}

```

这样再执行 php artisan migrate 就不会出错了

php artisan make:mode Mode/Admin -m 创建Admin模型，同进创建表migration  
迁移  
UserFactory.php 增加下面的代码，增加模拟数据的模型

```

$factory->define(App\Mode\Admin::class, function (Faker $faker) {
 static $password;

 return [
 'username' => $faker->name,
 'password' => $password ?: $password = bcrypt('admin'),
];
});

```

使用tinker来生成模拟数据

```

appledeiMac% php artisan tinker
Psy Shell v0.8.14 (PHP 7.1.11 — cli) by Justin Hileman
>>> factory(App\Mode\Admin::class,3)->create();
=> Illuminate\Database\Eloquent\Collection {#801
 all: [
 App\Mode\Admin {#797
 username: "Clarabelle Thiel",
 password:
 "$2y$10$KTpVr.SAw7OhLYfA46nixeyrwORpOpj3qP8kvqObtBXIGbuwOtQDq",
 }
]
}

```

```
updated_at: "2017-11-07 08:50:53",
created_at: "2017-11-07 08:50:53",
id: 1,
},
App\Mode\Admin {#793
 username: "Daphney Walker PhD",
 password:
"$2y$10$KTpVr.SAw7OhLYfA46nixeyrwORpOpj3qP8kvqObtBXIGbuwOtQDq",
 updated_at: "2017-11-07 08:50:53",
 created_at: "2017-11-07 08:50:53",
 id: 2,
},
App\Mode\Admin {#792
 username: "Maynard Mueller",
 password:
"$2y$10$KTpVr.SAw7OhLYfA46nixeyrwORpOpj3qP8kvqObtBXIGbuwOtQDq",
 updated_at: "2017-11-07 08:50:53",
 created_at: "2017-11-07 08:50:53",
 id: 3,
},
],
}
>>>
```

## 创建后台入口控制器

```
php artisan make:controller Admin/EntryController
```

```
//设置路由前缀admin, namespace指定命名空间
Route::group(['prefix'=>'admin','namespace'=>'admin'],function(){
 Route::get('/login','EntryController@loginForm');
});
```

```
Route::get('/login','Admin\EntryController@loginForm'); 同上面语句一样
```

## //后盾网的js

安装hdjs : npm install hdjs

安装完后会在根目录下会生成一个 node\_modules 的目录，  
接下来把 node\_modules 移动到 public 目录中

# Laravel&VueJs 开发移动与桌面平台视频播放项目

希望大家为本项目加个 Star，给我们提供继续坚持录制高质量视频的动力。

## 开发环境

## 开发工具

mac+phpstorm+sequelPro

## 程序语言

laravel+mysql+php+bootstrap+require.js+阿里云oss+阿里云  
sms+axios+vuejs+vuex...

## 插件

### ideHelper

是一个帮助提高laravel代码提示功能的插件

#安装插件

```
composer require barryvdh/laravel-ide-helper
```

#在 config/app.php 文件中的providers 配置段中添加以下  
Barryvdh\LaravelIdeHelper\IdeHelperServiceProvider::class

#生成辅助文件

```
php artisan ide-helper:generate
```

### laracasts/flash

laracasts/flash用于控制消息显示处理

#安装

```
composer require laracasts/flash
```

#在 config/app.php 文件中的providers 配置段中添加以下

```
'providers' => [
 Laracasts\Flash\FlashServiceProvider::class,
];
```

#生成模板

```
php artisan vendor:publish --provider="Laracasts\Flash\FlashServiceProvider"
```

## 模板中定义

```
@include('flash::message')
<script>
 $('#flash-overlay-modal').modal();
</script>
```

## 解决mysql5.7以下出错的问题

在 app\Providers\AppServiceProvider.php 文件里的 boot 方法里设置一个默认值

```
public function boot(){
 Schema::defaultStringLength(191);
```

}

## 解决ajax跨域访问

默认情况下前台发送Ajax是允许跨域请求的。我们可以在后台进行相关设置然后允许前台跨域请求。

允许单个域名访问

```
header('Access-Control-Allow-Origin:http://www.houdunwang.com');
允许多个域名 $origin = isset($_SERVER['HTTP_ORIGIN'])?
$_SERVER['HTTP_ORIGIN'] : '';
$allow_origin = array(
 'http://www.houdunren.com',
 'http://www.houdunwang.com'
,
if(in_array($origin, $allow_origin)){
 header('Access-Control-Allow-Origin:'.$origin);
}
允许所有域名请求 header('Access-Control-Allow-Origin:*');
```

## phpstorm安装bootstrap插件（让你快速布局的利器）

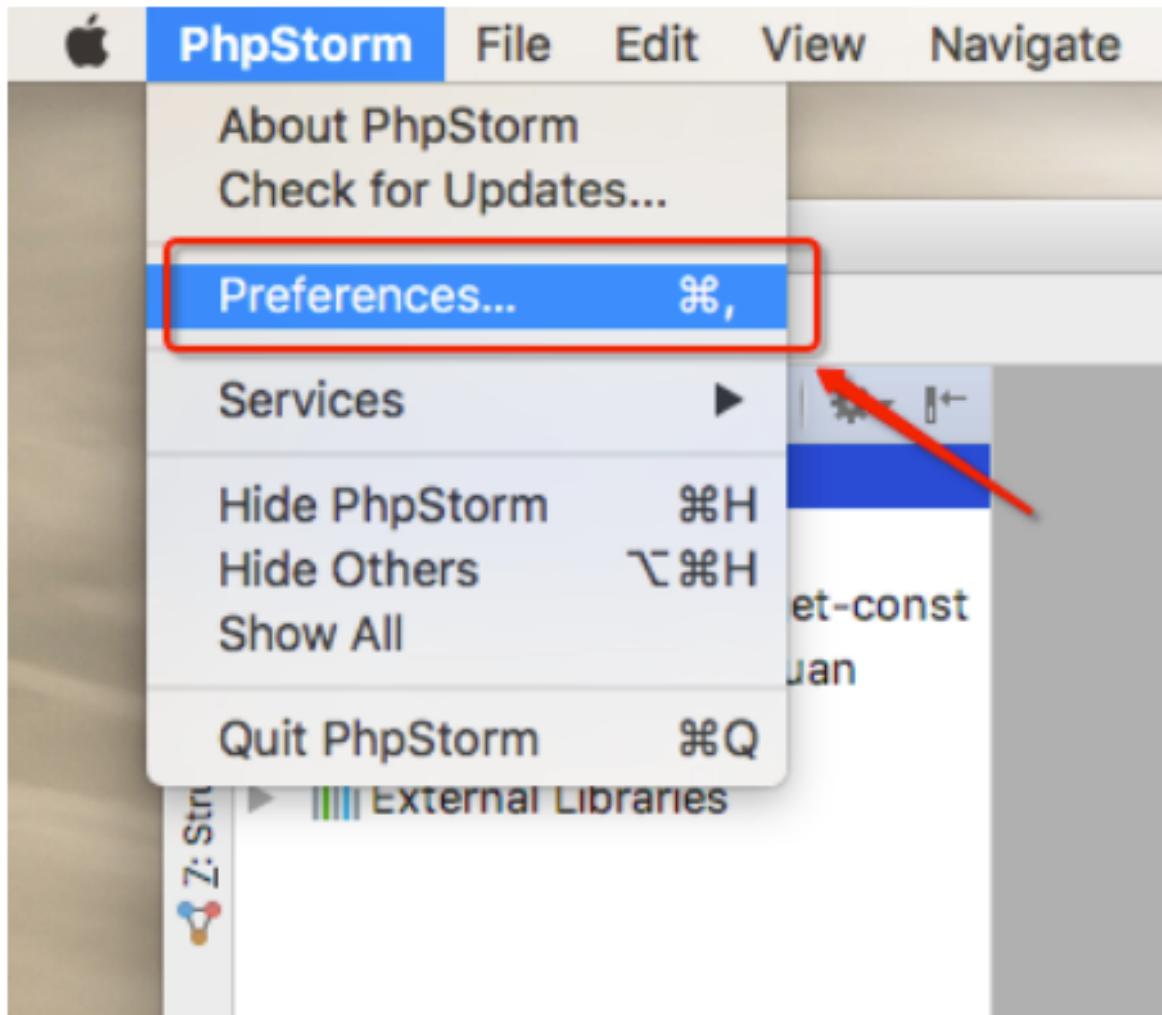
2017年6月29日 nick技术笔记

phpstorm是一款非常强大的编辑器，它有很多出色的功能，当然，它也提供了可扩展的功能（牛x的软件都应有此功能）。

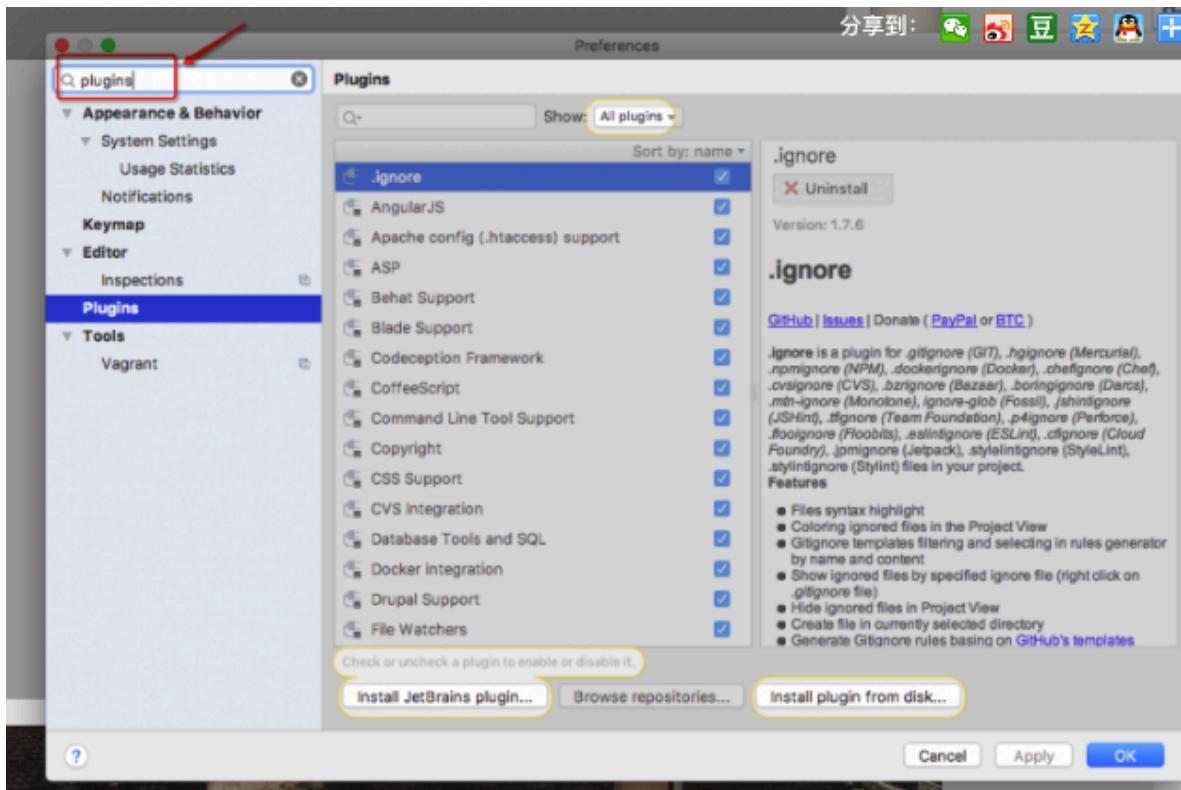
本文给大家说说如何让你的phpstorm变的更加强大，我们来安装bootstrap插件(后面我就用bt代替bootstrap了)，让你可以更加快速的使用bt布局页面。

当然大家不只通过本文学会了bt的安装，而是学会了插件的安装。

**1. 找到设置(preferences)， windows用户也如此，在工具栏可以很容易找到。**

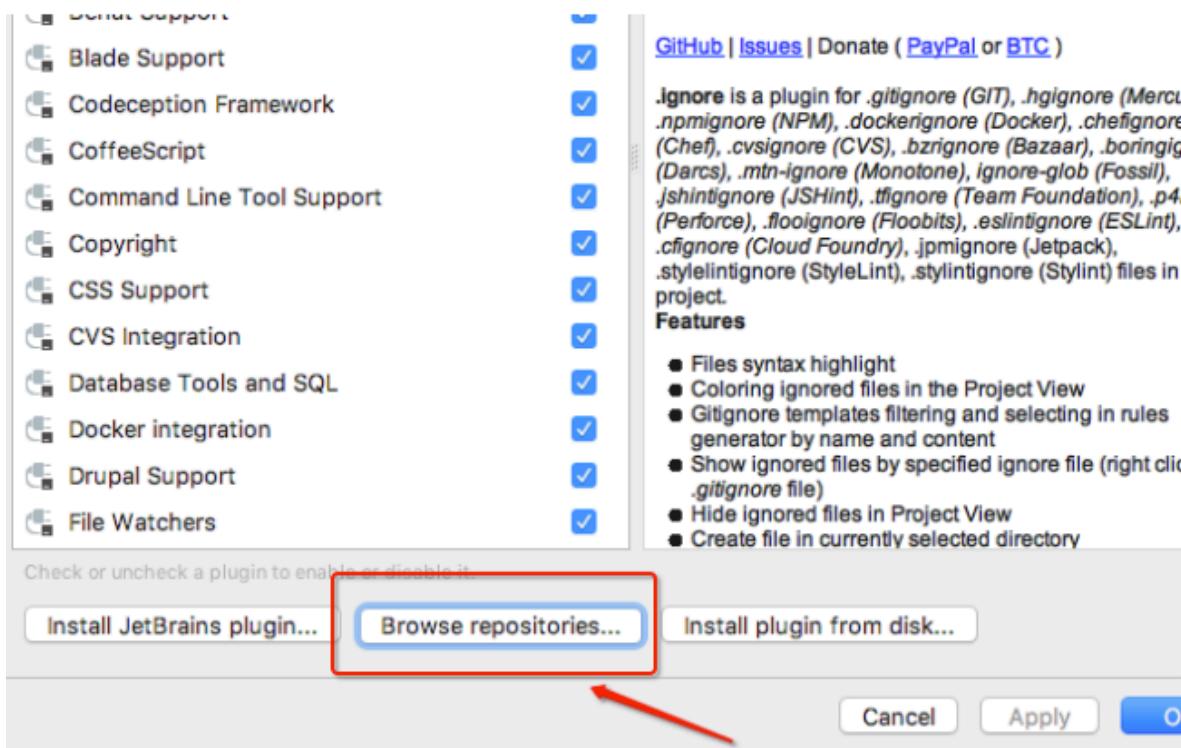


2.在输入框输入plugins (如果你汉化过了, 搜索插件)



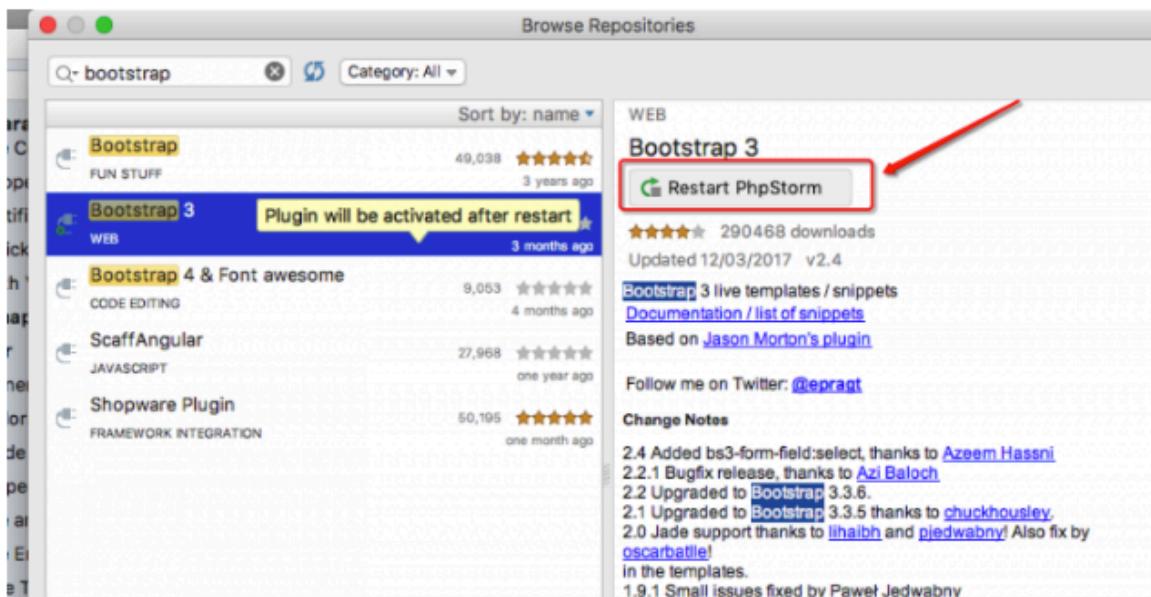
### 3. 安装

点击浏览仓库



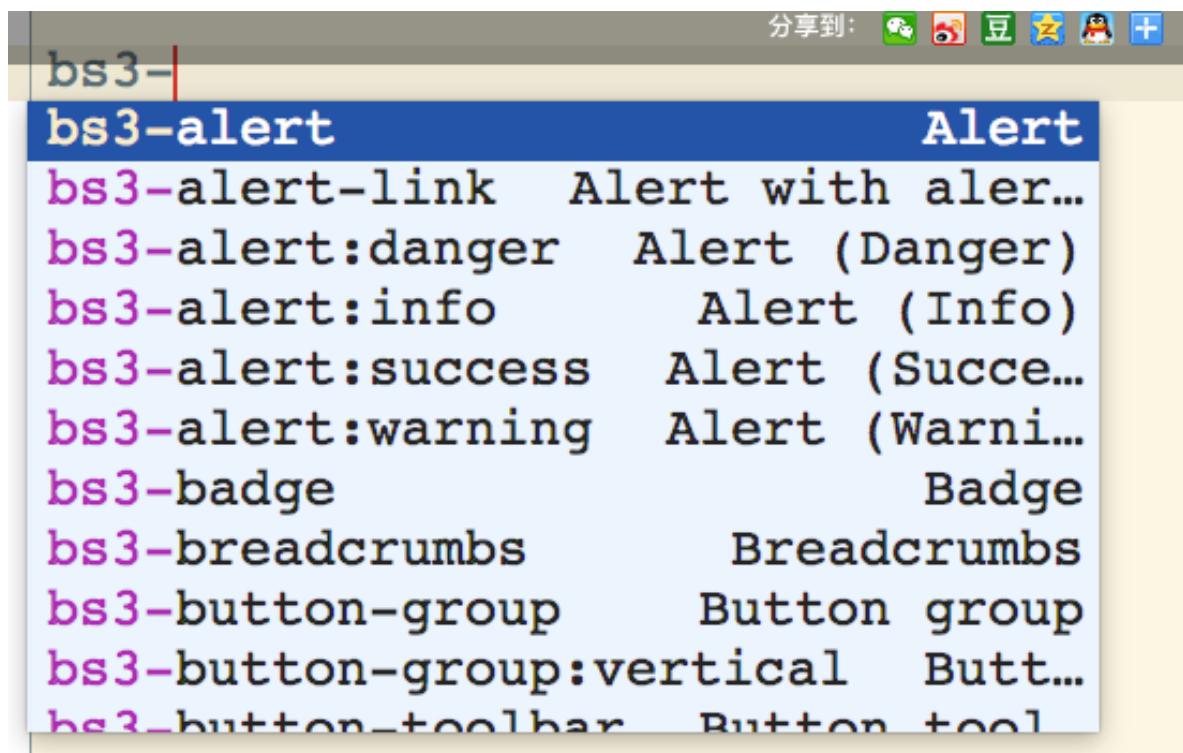
目前用的比较多的是3，所以安装bootstrap 3，不要安装错了

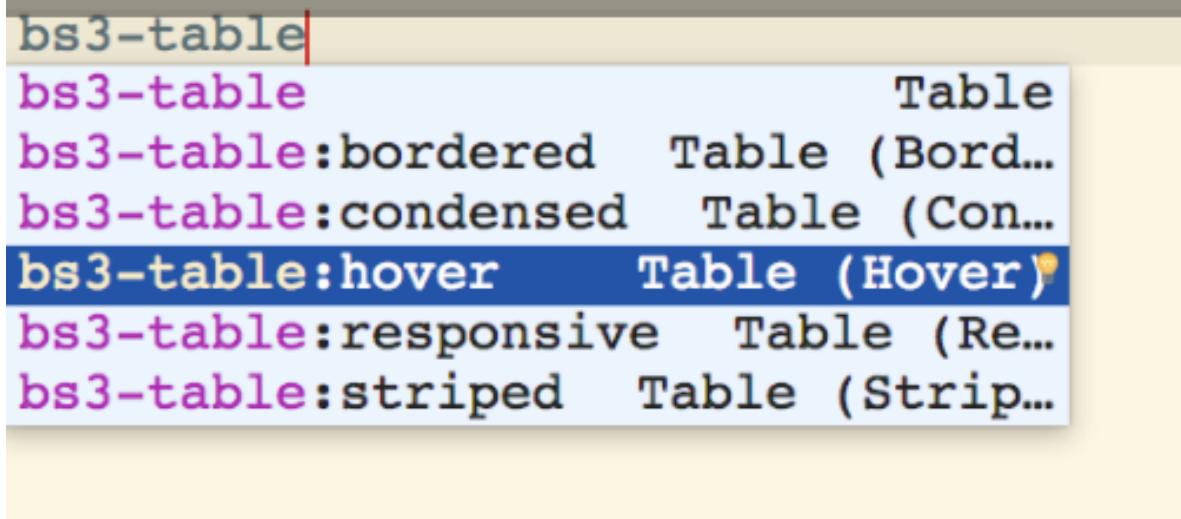
安装如果网速快，那么很快的，否则请耐心等一会儿，安装完毕之后点击Restart PhpStorm，重启phpstrom



## 4. 使用

新建或者打开一个文件，输入bs3-，然后按下ctrl+j，就可以使用了！





后记：好多人向我反映说phpstorm简直启动太慢了，还有某某操作不方便... ...，大家要知道万事万物都有优点和缺点，我们选择一种事和物，不是因为它十全十美，而是因为它优点要多于缺点。再一个，不是phpstorm启动太慢，是你的电脑配置太低，正所谓：工欲善其事，必先利其器，所以，你懂得。

<http://www.nickblog.cn/wp-content/uploads/2017/09/%E6%B5%81%E7%A8%8B.html>  
<http://www.nickblog.cn/category/category技术笔记>

Phpstrom生成多行注释  
ctrl+shift+/-

或者先敲 `/**然后回车`

- - [laravel框架实战](#)
    - [手册](#)
    - ◆ [安装与配置](#)
      - ◆ [安装框架](#)
      - ◆ [安装提示插件](#)
      - ◆ [配置数据库](#)
      - ◆ [数据迁移](#)
      - [远程配置](#)
    - ◆ [项目开发-后台](#)
      - ◆ [模型/迁移/填充操作\(以后台用户表为例\)](#)
      - ◆ [控制器处理](#)
      - ◆ [路由处理](#)
      - ◆ [定义自己的函数和类](#)
      - ◆ [视图处理](#)
      - ◆ [登陆处理](#)
      - ◆ [后台视图处理](#)
      - ◆ [退出处理](#)
      - ◆ [获取用户信息](#)
      - ◆ [更改密码](#)
        - ◆ [表单验证](#)
        - ◆ [消息提示](#)
      - ◆ [内容标签](#)
        - ◆ [创建资源控制器](#)
        - ◆ [添加](#)
        - ◆ [列表](#)
        - ◆ [删除操作](#)
        - ◆ [编辑操作](#)
      - ◆ [公共继承](#)
      - ◆ [课程视频管理](#)
        - ◆ [上传](#)
        - ◆ [课程添加](#)
        - ◆ [OSS上传](#)
        - ◆ [编辑](#)
        - [删除操作](#)
      - ◆ [项目开发-前台](#)
        - ◆ [首页](#)
        - ◆ [列表页](#)
        - ◆ [内容页](#)
        - ◆ [增加router-link完成跳转](#)
        - [vue+swiper使用](#)
      - ◆ [构建接口](#)
        - ◆ [标签接口](#)
        - ◆ [课程接口](#)
        - ◆ [推荐与热门接口](#)
        - [视频接口](#)

- ◆ [安装axios调用接口数据](#)
- ◆ [解决ajax跨域访问](#)
- ◆ [推荐热门课程接口调用](#)
- ◆ [视频Page的接口调用](#)
- ◆ [获取标签](#)
- ◆ [过渡动画效果](#)
- ◆ [编译](#)
- ◆ [打包APP](#)

## laravel框架实战

### 手册

<http://d.laravel-china.org/docs/5.4>

### 安装与配置

#### 安装框架

配置**composer.json**, 修改为中国镜像,或者全局配置好

```
"repositories": {
 "packagist": {
 "type": "composer",
 "url": "https://packagist.phpcomposer.com"
 }
}
```

提前配置好国内镜像, 否则很慢...

```
composer create-project --prefer-dist laravel/laravel video
```

#### 安装提示插件

```
#打开packagist.org搜索 laravel-ide-helper
```

```
#提前配置好中国镜像
```

```
composer require barryvdh/laravel-ide-helper
```

#在config/app.php中, 找到providers在最后一行增加配置项  
Barryvdh\ Laravel\ IdeHelper\ IdeHelperServiceProvider::class,

#然后再执行

```
php artisan ide-helper:generate
```

#重启ide编辑器, 找到有一个路由文件Config::试试

#### 配置数据库

**mysql版本最好是5.7, 找到.env, 然后修改配置。**

#### 数据迁移

#创建表的数据迁移文件

```
php artisan make:migration create_users_table --create=users
```

#执行迁移,如果版本过低就会报错

```
php artisan migrate
```

**更改config/database.php,**

把

```
'charset' => 'utf8mb4',
'collation' => 'utf8mb4_unicode_ci',
```

修改成

```
'charset' => 'utf8',
'collation' => 'utf8_unicode_ci',
```

就可以支持mysql低版本了

## 1. 远程配置

2. 项目传到服务器，使用phpstorm同步  
    配置远程数据库，使用工具连接

项目开发-后台

### • 模型/迁移/填充操作(以后台用户表为例)

模型配置

执行以下命令创建模型，如果加上 -m 代表一同创建数据迁移文件。

创建完毕之后在app目录会有一个**Model**目录并里面就会有模型。

**database/migrations**也会多一个数据迁移文件。

### • php artisan make:model Model/Admin -m

数据迁移

默认的**users**作为前台用户表，我们使用**admins**作为后台用户表，在迁移文件中，增加**admins**字段(**username** 非重， **password**)

执行数据迁移

### • php artisan migrate

数据填充

找到**database/factories/ModelFactory.php**文件

复制已有代码改成

```
$factory->define(\App\Model\Admin::class, function (Faker\Generator $faker) {
 static $password;

 return [
 'username' => $faker->name,
 'password' => $password ?: $password = bcrypt('admin888'),
];
});
```

然后在命令行执行**tinker**命令来灌数据

```
php artisan tinker
```

在命令行输入， 创建3条用户表的数据

```
factory(App\Model\Admin::class,3)->create();
```

控制器处理

创建控制器，会在**app/Http/Controllers**生成控制器

```
php artisan make:controller Admin/EntryController
```

路由处理

在控制器中建立一个**login**方法，然后需要设置路由访问，  
不能直接通过get参数的形式访问。**laravel 3** 有， **laravel 4** 取消了这样

的模式

找到**routes/web.php**建立路由

```
Route::get('/login', 'Admin\EntryController@login');
```

我们也可以把路由分组，路由组允许共享路由属性，例如中间件和命名空间等，我们没有必要为每个路由单独设置共有属性，共有属性会以数组的形式放到 **Route::group** 方法的第一个参数中。

```
Route::group(['prefix'=>'admin','namespace'=>'Admin'],
function(){
 //prefix是admin，所以匹配包含 "/admin/login" 的 URL
 //namespace为Admin，会自动寻找App\Http\Controllers\Admin下面的
 EntryController
 Route::get('/login','EntryController@login')
});
```

也可以在**web.php**写入，把路由按照文件夹分开

```
include __DIR__ . '/admin/web.php';
```

如果**get**和**post**同时请求一个方法可以使用**match**

```
Route::match(['get','post'],'/login', 'EntryController@login');
```

定义自己的函数和类

在**app**目录下面建立**Libraries**目录，建立**helper.php**写入p函数

更改**composer.json**

```
"files": [
 "app/Libraries/helper.php"
]
```

还要执行 **composer dump**

视图处理

默认视图位置**resource/views/**，载入模板可以通过

```
return view('welcome');
```

也可以

```
return \View::make('welcome');
```

如果要建立文件夹**views/admin/entry/login.blade.php**，那么需要：

```
return view('admin.entry.login');
```

把**hdcms**的后台模板另存作为我们项目所用，静态资源放入到**public**目录

表单提交需要在**form**元素中加

```
{{csrf_field()}}
```

登陆处理

使用**laravel**提供的用户验证

更改**config/auth.php**文件

```
//增加admin守卫(guards)，驱动为session，交给提供者(provider)的admins处理，然后再找到App\Model\Admin模型，模型需要继承
```

```
Illuminate\Foundation\Auth\User
'guards' => [
 'admin' => [
 'driver' => 'session',
 'provider' => 'admins',
],
 'web' => [
 'driver' => 'session',
 'provider' => 'users',
],
],
'api' => [
 'driver' => 'token',
 'provider' => 'users',
],
],
'providers' => [
 'users' => [
 'driver' => 'eloquent',
 'model' => App\User::class,
],
 'admins' => [
 'driver' => 'eloquent',
 'model' => App\Model\Admin::class,
]
],
],
```

## Admin模型

```
<?php
```

```
namespace App\Model;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Foundation\Auth\User as Authenticatable;
class Admin extends Authenticatable
{
 //
}
```

## 在控制器

```
$status = Auth::guard('admin')->attempt([
 'username'=>Request::input('username'),
 'password'=>Request::input('password'),
]);
```

## 如果登陆成功或者登陆失败

```
if($status){
 //跳转到后台首页，注意写路由
 return redirect('/admin/index');
}
//重新跳转回来，并且分配闪存属性
return redirect('/admin/login')->with('error','用户名或者密码错误');
```

## 页面代码：

```
@if(session('error'))
 <div class="alert alert-danger">
 {{session('error')}}
 </div>
@endif
```

## 通过中间件来验证登陆

php artisan make:middleware AdminMiddleware

### 在中间件中写入检测是否登陆

```
if(!Auth::guard('admin')->check()){
 return redirect('/admin/login');
}
```

## 写入配置项app/Http/Kernel.php的\$routeMiddleware

'admin.auth'=>AdminMiddleware::class

### 在Entry控制器构造方法调用中间件，除了登陆方法

```
public function __construct() {
 $this->middleware('admin.auth')->except(['login']);
}
```

## 后台视图处理

### 模板继承：

**resources/views/admin/entry新建index.blade.php**，让后台首页方法载入

```
return view('admin.entry.index')
```

然后建立**resources/views/admin/layout**文件夹放入继承的父模板**master.blade.php**找到不是共用的部分，写入占位符

```
@yield('content')
```

### 在index.blade.php写入以下代码

```
@extends('admin.layout.master')
@section('content')
中间内容
@endsection
```

## 退出处理

### 构建控制器方法logout

```
Auth::guard('admin')->logout();
return redirect('/admin/login');
```

## 更改令牌app/Model/Admin.php

```
protected $rememberTokenName = '';
```

### 获取用户信息

```
//html页面
{Auth::guard('admin')->user()->username}}
```

### 更改密码

创建控制器，配置路由指向控制器方法**changePassword**，更改父模板a链接，构建修改密码页面

```
php artisan make:controller Admin/MyController
public function password(){
 return view('admin.my.password');
}
```

在**password.blade.php**页面的**form**表单指向到/**admin/changePassword**，别忘记设计路由

### 表单验证

创建表单验证，在**app/Http/Requests**会有**PasswordPost.php**文件

```
php artisan make:request PasswordPost
```

### authorize方法

表单的请求类内包含了**authorize**方法。在这个方法中，你可以确认用户是否真的通过了授权，以便更新指定数据，比如是否登陆？如果自己处理验证，那么返回**true**

```
public function authorize()
{
 return true;
}
```

### rules方法是规则，

#### confirmme

验证字段值必须和**foo\_confirmation**的字段值一致。例如，如果要验证的字段是**password**，就必须和输入数据里的**password\_confirmation**的值保持一致。

```
public function rules() {
 return [
 'old_password' => 'sometimes|required',
 'password' => 'sometimes|required|confirmed',
 'password_confirmation' => 'sometimes|required',
];
}
```

### 在**password.blade.php**页面显示错误消息

```
@if (count($errors) > 0)
 <div class="alert alert-danger">

```

```
@foreach ($errors->all() as $error)
 {{ $error }}
@endforeach

</div>
@endif
```

**在app/Http/Controllers/Admin/MyController::changePassword方法中，注入验证对象**

```
public function changePassword(PasswordPost $request){
 $model = Auth::guard('admin')->user();
 $model->password = bcrypt($request->input('password'));
 $model->save();
}
```

改成汉语的错误提示：

**在app/Http/Requests/PasswordPost.php**你可以通过重写表单请求的 messages 方法来自定义错误消息。此方法必须返回一个数组，其中含有成对的属性或规则以及对应的错误消息

```
public function messages()
{
 return [
 'old_password.required' => '原始密码不能为空',
 'password.required' => '新密码不能为空',
 'password.confirmed' => '两次密码不一致',
 'password_confirmation.required' => '确认密码不能为空'
];
}
```

**验证原密码是否正确，新建方法passwordValidator，然后在rules调用**

```
use Validator;
use Hash;
public function passwordValidator(){
 Validator::extend('check_password', function ($attribute, $value,
 $parameters, $validator) {
 return Hash::check($value,Auth::guard('admin')->user()->password);
 });
}
```

**rule调用，并且制定check\_password规则，别忘记在messages增加提示信息**

```
public function rules() {
 $this->passwordValidator();
 return [
 'old_password' => 'sometimes|required|check_password',
 'password' => 'sometimes|required|confirmed',
```

```
'password_confirmation' => 'sometimes|required',
];
}
```

## 消息提示

在packagist.org搜索flash组件

composer require laracasts/flash

把以下代码放在**config/app.php**的**provider**配置下

Laracasts\Flash\FlashServiceProvider::class,

在**master.blade.php**文件尾部写入

```
@include('flash::message')
<script>
require(['bootstrap'],function($){
 $('#flash-overlay-modal').modal();
 setTimeout(function(){
 $('#flash-overlay-modal').modal('hide');

 },2000)
});
</script>
```

生成消息提示模板，也可以直接更改模板，会在**resources/views**生成**vendor**目录

php artisan vendor:publish --provider="Laracasts\Flash\FlashServiceProvider"

在控制器方法中使用**flash**方法

```
public function changePassword(PasswordPost $request){
 if($request->isMethod('post')){
 $model = Auth::guard('admin')->user();
 $model->password = bcrypt($request->input('password'));
 $model->save();
 flash()->overlay('密码修改成功', '后盾人-友情提示');
 }
 return view('admin.my.changePassword');
}
```

## 内容标签

创建资源控制器

**Laravel** 资源路由可以将典型的「CURD」路由指定到一个控制器上

php artisan make:controller Admin\\TagController --resource

设置路由器

```
Route::resource('tag','TagController');
GET => /tag => index方法
GET => /tag/create => create方法
POST => /tag => store方法
GET => /tag/{id} => show方法
GET => /tag/{id}/edit => edit方法
```

...

## 添加

构建标签列表和标签添加模板，让各个方法**create**、**index**载入模板。

创建标签模型，顺便创建表

```
php artisan make:model Model/Tag -m
```

**数据迁移文件中创建字段name**

```
Schema::create('tags', function (Blueprint $table) {
 $table->increments('id');
 $table->string('name',50);
 $table->timestamps();
});
```

**post**请求就会到**store**方法，**form**表单写法

```
<form action="/admin/tag" method="post" ...
```

创建表单验证，会在**app\Http\Requests**目录生成表单验证类，按照**passwordPost.php**修改就好了。

```
php artisan make:request TagPost
```

更改控制器中的**store**方法注入

```
public function store(TagPost $request)
{
 if($request->isMethod('post')){
 p($request->input());
 }
}
```

别忘记在模板添加错误提示

```
@if (count($errors) > 0)
 <div class="alert alert-danger">

 @foreach ($errors->all() as $error)
 {{ $error }}
 @endforeach

 </div>
@endif
```

**批量填充， 默认不允许**

```
public function store(TagPost $request, Tag $model)
{
 if($request->isMethod('post')){
 $model->create($request->all());
 flash()->overlay('添加成功','友情提示');
 }
}
```

需要在**App\Model\Tag**模型中指定**guarded = []**

```
class Tag extends Model
{
```

```
//guarded警戒的意思，为空代表谁也不警戒
protected $guarded = [];
}

添加成功提示
public function store(TagPost $request, Tag $model)
{
 if($request->isMethod('post')){
 $model->create($request->all());
 flash()->overlay('添加成功','友情提示');
 return view('admin.tag.create');
 }
}
```

## 列表

### 列表页处理

```
public function index(Tag $model)
{
 $data = $model->get();
 return view('admin.tag.index',compact('data'));
}
```

### 页面foreach循环

```
@foreach($data as $v)
<tr>
 <td>{{$v['id']}}</td>
 <td>{{$v['name']}}</td>
 <td>
 <div class="btn-group">
 编辑

 删除

 </div>
 </td>
</tr>
```

```
@endforeach
```

### 删除操作

#### 书写js的del方法

```
<script>
 function del(id){
 require(['util'],function(util){
 util.confirm('确定删除吗？ ',function(){
 $.ajax({
 url:'/admin/tag/' + id,
 method:'DELETE',
 success:function(response){

```

```
 })
 });
}
</script>
@endsection
```

**但是监控异步发现需要使用csrf\_token，可以全局异步设置csrf\_token传输，在master.blade.php模板中设置以下代码**

```
<meta name="csrf-token" content="{{ csrf_token() }}>
```

```
<script>
//放在require.js下面
require(['jquery'],function($){
$.ajaxSetup({
headers: {
'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
}
});
})
</script>
```

## **控制器destroy方法操作**

```
public function destroy($id)
{
Tag::destroy($id);
return response()->json(['message'=>'删除成功','status'=>true]);
}
```

## **异步的success方法**

```
success:function(response){
 util.message(response.message,'refresh');
}
```

## **编辑操作**

### **edit方法**

```
public function edit($id)
{
 $model = Tag::find($id);
 return view('admin.tag.edit',compact('model'));
}
```

## **模板**

```
<input type="text" name="name" id="inputID" class="form-control"
value="{{ $model['name'] }}" title="">
```

**修改动作，需要模拟put请求，action需要带上参数，才能访问update方法**

```
<form action="/admin/tag/{{ $model['id'] }}"/>....
```

```
 {{method_field('PUT')}}
```

## update方法

```
public function update(Request $request, $id)
{
 //
 $model = Tag::find($id);
 $model['name'] = $request->input('name');
 $model->save();
 return redirect('/admin/tag');
}
```

## 公共继承

建立公共控制器，通过建立构造方法，然后让自控制器继承，来完成验证，入口控制器不继承了，需要排除一些方法。

```
php artisan make:controller Admin/CommonController
```

## 课程视频管理

### 创建控制器

```
php artisan make:controller Admin/LessonController
```

### 配置资源路由

```
Route::resource('lesson','LessonController');
```

创建模型，顺便创建数据迁移文件，别忘记执行数据迁移。

```
php artisan make:model Model/Lesson -m
```

### 更改迁移文件，增加字段

```
Schema::create('lessons', function (Blueprint $table) {
 $table->increments('id');
 $table->string('title');
 $table->string('introduce');
 $table->string('preview');
 $table->tinyInteger('iscommend');
 $table->tinyInteger('ishot');
 $table->smallInteger('click');
 $table->timestamps();
});
```

## 执行迁移

```
php artisan migrate
```

模板处理，直接复制tag，然后修改。

## 在课程下面使用Vue处理视频

```
<script>
 require(['vue'],function(Vue){
 new Vue({
 el:'#app',
 data:{
 videos:[{title:"",path:""}]
```

```

 },
 methods:{
 add(){
 this.videos.push({title:"",path:""});
 },
 del(k){
 this.videos.splice(k,1);
 }
 }
})
})

```

</script>

页面可以查看数据，注意加上@符号，代表laravel不要以模板引擎方法来解析，因为这是vue模板引擎。

@{{videos}}

## 创建视频video模型与表

php artisan make:model Model/Video -m

## 数据迁移文件

```

Schema::create('videos', function (Blueprint $table) {
 $table->increments('id');
 $table->string('title');
 $table->string('path');
 $table->integer('lesson_id');
 $table->timestamps();
});

```

## 执行迁移

php artisan migrate

## 更改Model/Video模型，允许批量填充

protected \$guarded = [];

## 在Model/Lesson模型中，声明多表关联，也就是一门课程拥有多个视频

//允许批量填充

protected \$guarded = [];

public function videos(){

//videos表和lessons表关联自动，会自动以当前模型(lesson)的名称加上\_id，也就是lesson\_id就是video表的关联字段【Snake Case】

return \$this->hasMany(Video::class);

}

## 手动在表中插入一些模拟数据，然后测试：

```

$data = Lesson::find(1)->videos()->get();
$p($data->toArray());

```

- 上传

fileinfo扩展安装

上传需要用到**fileinfo**扩展，需要在宝塔后台安装：**fileinfo**，如果安装

- 失败，在Linux工具箱，把**swap**交换分区调整成1024M

使用hdjs插件完成上传

将以下代码放在课程添加页面，别忘记改**name**名称

```
<div class="col-sm-11">
 <div class="input-group">
 <input type="text" class="form-control" name="preview" readonly="" value="">
 <div class="input-group-btn">
 <button onclick="uplImage(this)" class="btn btn-default" type="button">选择图片</button>
 </div>
 </div>
 <div class="input-group" style="margin-top:5px;">

 <em class="close" style="position: absolute; top: 0px; right: -14px;" title="删除这张图片" onclick="removelmg(this)">x
 </div>
</div>
<script>
 //上传图片
 function uplImage(obj) {
 require(['util'], function (util) {
 options = {
 multiple: false,//是否允许多图上传
 };
 util.image(function (images) { //上传成功的图片，数组类型
 $("[name='preview']").val(images[0]);
 $(".img-thumbnail").attr('src', images[0]);
 }, options)
 });
 }

 //移除图片
 function removelmg(obj) {
 $(obj).prev('img').attr('src', '{{asset('images/nopic.jpg')}}');
 $(obj).parent().prev().find('input').val("");
 }
</script>
```

图片地址可以是

```


```
 [
    'driver' => 'local',
    //'root' => storage_path('app'),
    'root' => 'attachment',
],
```

建立后台控制器配合hdjs上传

```
php artisan make:controller Component/UploadController
```

建立**uploader**和**filesLists**方法：

```
//上传文件
```

```
public function uploader( Request $request ) {
```

```
    //
    $upload = $request->file;
    //可以查看$upload的方法，比如获得大小，类型等...
    // $arr = get_class_methods($upload);
    // p($arr);
```

```
//判断是否上传成功
```

```
if($upload->isValid()) {
```

```
    $path = $upload->store(date('ymd'));
    return ['valid'=>1,'message'=>asset('attachment/' . $path)];
}
```

```
return ['valid'=>0,'message'=>'上传失败'];
```

```
}
```

配置路由，不属于后台，直接在**routes/web.php**配置就好了

```
Route::match(['get', 'post'], '/component/uploader',
    'Component(UploadController@uploader');
```

```
Route::match(['get', 'post'], '/component/filesLists',
    'Component(UploadController@uploader');
```

更改**master.blade.php**中的hdjs的配置

```
//HDJS组件需要的配置
```

```
hdjs = {
    'base': '/node_modules/hdjs',
    'uploader': '/component/uploader',
    //后面系统需要传参数，所以加一个问号
```

```
'filesLists': '/component/filesLists?',
'removeImage': '?s=component/upload/removeImage&m=teacher&siteid=11',
'ossSign': '?s=component/oss/sign&m=teacher&siteid=11',
};
```

创建标签与课程的中间表

```
php artisan make:model Model/LessonTag -m
```

设置模型

```
protected $guarded = [];
```

数据迁移

```
Schema::create('lesson_tags', function (Blueprint $table) {
    $table->integer('lesson_id');
    $table->integer('tag_id');
    $table->timestamps();
});
```

```
php artisan migrate
```

课程添加

在LessonController中的store方法添加数据

```
public function store( Request $request ) {
```

```
//添加课程
```

```
$lesson      = new Lesson();
$lesson->title    = $request->input( 'title' );
$lesson->introduce = $request->input( 'introduce' );
$lesson->preview   = $request->input( 'preview' );
$lesson->iscommend = $request->input( 'iscommend' );
$lesson->ishot     = $request->input( 'ishot' );
$lesson->click     = $request->input( 'click' );
$lesson->save();
```

```
//添加中间表
```

```
foreach ( $request->input( 'tid' ) as $tid ) {
```

```
//添加标签与课程的中间表
```

```
( new LessonTag() )->create(
```

```
[
```

```
    'lesson_id' => $lesson->id,
    'tag_id'    => $tid
]
```

```
);
```

```
}
```

```
//添加视频
```

```
foreach ( json_decode( $request->videos, true ) as $v ) {
```

```

$lesson->videos()->create(
    [
        'title'=>$v['title'],
        'path'=>$v['path']
    ]
);
}

return redirect( '/admin/lesson' );
}

```

OSS上传

新建oss, 进入到

<https://oss.console.aliyun.com/overview>

开通阿里云的oss,

- 友情提示：在阿里云存入1-2元钱，以免播放没钱，不能播放了。
 - 新建块Bucket, 选择公共读
基础设置->跨域设置->创建规则->来源为*,Allowed允许所有请求， Allowed
 - Headers: 也设置为*
 - 鼠标移动右上角的用户名，点击访问控制
到访问控制控制台，点击用户管理，然后新建用户，勾选下面的“为改用自动生
 - 成AccessKey”,然后保存AccessKey
 - 找到用户授权,让其拥有操作OSS的权限(管理开放存储服务(OSS)权限)
再次进入OSS控制台，找到外网域名和块的名称复制保存。

在组件中新建控制器

php artisan make:controller Component/OssController

打开HDJS手册

<http://hdjs.hdphp.com/357757>

复制手册中的sign方法到OssController中，注意把static去掉，然后设置信息。

配置路由

Route::any('/component/oss', 'Component\OssController@sign');

复制以下代码页面做测试

```

<div class="form-group">
    <label for="inputID" class="col-sm-2 control-label">视频地址:</label>
    <div class="col-sm-10">
        <div class="input-group">
            <input type="text" class="form-control" v-model="v.path">
            <span class="input-group-btn">
                <button class="btn btn-default" type="button" :id="v.id">上传视频</button>
            </span>
        </div>
        <b style="margin-top: 3px;color: red" hidden :id="'process'+v.id">0%</b>
    </div>
</div>

```

```
<script>
require(['vue'], function (Vue) {
new Vue({
  el: '#app',
  data: {
    videos: []
  },
  methods: {
    add() {
      var field = {title: "", path: "", id: 'hd' + Date.parse(new Date())};
      this.videos.push(field);
      setTimeout(function () {
        upload(field);
      }, 200);
    },
    del(k) {
      this.videos.splice(k, 1);
    }
  }
})
})

function upload(field) {
  require(['oss'], function (oss) {
    var id = '#' + field.id;
    var uploader = oss.upload({
      //获取签名
      serverUrl: '/component/oss?',
      //上传目录
      dir: 'houdunwang/',
      //按钮元素
      pick: id,
      accept: {
        title: 'video',
        extensions: 'mp4',
        mimeTypes: 'video/mp4'
      }
    });
    //上传开始
    uploader.on('startUpload', function () {
      console.log('开始上传');
    });
    //上传成功
    uploader.on('uploadSuccess', function (file, response) {
      field.path = oss.oss.host + '/' + oss.oss.object_name;
      console.log('上传完成,文件名:' + oss.oss.host + '/' +
```

```

        oss.oss.object_name);
    });
    //上传中
    uploader.on('uploadProgress', function (file, percentage) {
        $('#process' + field.id).show().html(parseInt(percentage * 100) + '%');
        console.log('上传中,进度:' + parseInt(percentage * 100));
    })
    //上传结束
    uploader.on('uploadComplete', function () {
        $('#process' + field.id).hide()
        console.log('上传结束');
    })
});
}

</script>

```

编辑

代码如下

```

public function edit( $id ) {
    //
    $model = Lesson::find( $id );
    $videos = $model->videos()->get();
    $videos = json_encode( $videos, JSON_UNESCAPED_SLASHES |
JSON_UNESCAPED_UNICODE );

    return view( 'admin.lesson.edit', compact( 'model', 'videos' ) );
}

```

Vue代码

```

require(['vue'], function (Vue) {
    new Vue({
        el: '#app',
        data: {
            videos: JSON.parse('{!! $videos !!}')
        },
        mounted(){
            this.videos.forEach(function(v){
                upload(v);
            });
        },
        methods: {
            add() {
                var field = {title: "", path: "", id: 'hd' + Date.parse(new Date())};
                this.videos.push(field);
            }
        }
    });
})

```

```

        setTimeout(function () {
            upload(field);
        }, 200);
    },
    del(k) {
        this.videos.splice(k, 1);
    }
}
})
})

```

执行修改 html页面

```
<form action="/admin/lesson/{{\$model['id']}}" method="post" ....
{{method_field('PUT')}}
```

update方法

```

public function update( Request $request, $id ) {
    //添加课程
    $lesson      = Lesson::find($id);
    $lesson->title  = $request->input( 'title' );
    $lesson->introduce = $request->input( 'introduce' );
    $lesson->preview  = $request->input( 'preview' );
    $lesson->iscommend = $request->input( 'iscommend' );
    $lesson->ishot    = $request->input( 'ishot' );
    $lesson->click    = $request->input( 'click' );
    $lesson->save();
    //删除视频
    $lesson->videos()->delete();
    //添加视频
    foreach ( json_decode( $request->videos, true ) as $v ) {
        $lesson->videos()->create(
            [
                'title' => $v['title'],
                'path'  => $v['path']
            ]
        );
    }

    return redirect( '/admin/lesson' );
}

```

删除操作

```

public function destroy( $id ) {
    $model = Lesson::find( $id );
    //删除视频
    $model->videos()->delete();
}

```

```
//删除标签中间表
LessonTag::where(['lesson_id'=>$id])->delete();
//删除课程
$model->delete();

return [ 'message' => '删除成功', 'status' => true ];

}
```

项目开发-前台

利用**vue**的脚手架来创建一个**webapp**应用，按照**ppt**来执行几个命令，注意排除目录

运行脚手架

```
cd webapp
cnpm run dev
```

提取公共的**css**，放到**index.html**

```
<link rel="stylesheet" type="text/css" href="css/bootstrap.min.css" />
<link rel="stylesheet" type="text/css" href="css/iconfont.css"/>
```

把**css,js,image**s都放入到静态资源**static**目录

把链接指向**static**目录

```
<link rel="stylesheet" href="static/css/swiper-3.4.1.min.css" />
<link rel="stylesheet" type="text/css" href="static/css/bootstrap.min.css" />
<link rel="stylesheet" type="text/css" href="static/css/iconfont.css"/>
```

首页

复制一个**Hello.vue**改名为**Home.vue**，其中的**name**值就是在**vue**报错误的时候，能标明是哪个组件

```
<template></template>
<script>
export default {
  name: 'home',
  data () {
    return {
      }
    }
  }
</script>
<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped></style>
```

复制模板**index.html**的**body**中间的内容，放入到**Home.vue**中的**template**中，但是**template**中间不能有任何的Js代码。

设置路由

在**router/index.js**中

```
import Vue from 'vue'
import Router from 'vue-router'
```

```

import Home from '@/components/Home'

Vue.use(Router)

export default new Router({
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    }
  ]
})

```

设置首页的css，把index.css里面的代码复制到Home.vue中的style中，再把App.vue中的多余的元素删掉，“今日推荐”留一个就可以了。

- **列表页**

- 复制一个Home.vue作为Video.vue，
然后把模板video.html的body中间的内容复制到Video.vue中的template>div中
- 间去，注意不能有js代码，
- 然后把video.css的代码复制到Video.vue中的style中去
然后设置路由router/index.js，别忘记import js import Video from '@/components/Video' { path: '/video', name: 'Video', component: Video }
- 通过地址栏 /video来测试访问，注意替换图片路径

内容页

和上面一样，不再赘述

增加router-link完成跳转

其实就是把之前的a，替换成router-link，例如Page.vue：

```

<a href="" class="iconfont back">;</a>
//替换成
<router-link to="/" class="iconfont back">;</router-link>

```

- **vue+swiper使用**

去Npm搜索vue-awesome-swiper，或者直接打开(需要翻墙) <https://www.npmjs.com/package/vue-awesome-swiper> <https://github.com/surmon-china/vue-awesome-swiper>

安装插件

```

cd webapp
npm install vue-awesome-swiper --save

```

在main.js让vue扩展出swiper功能

```

import Vue from 'vue'
import VueAwesomeSwiper from 'vue-awesome-swiper'
Vue.use(VueAwesomeSwiper)

```

Home.vue的html代码

```

<!--轮播图-->
<swiper :options="swiperOption" :not-next-tick="notNextTick"

```

```

ref="mySwiper">
  <!-- slides -->
  <swiper-slide v-for="v in slide" :key="v.id">
    <router-link to="/video">
      
    </router-link>
  </swiper-slide>
  <!-- Optional controls -->
  <!-- Optional controls -->
  <div class="swiper-pagination" slot="pagination"></div>

</swiper>
<!--轮播图结束-->

```

然后在Home.vue页面的Js代码

```

export default {
  name: 'home',
  data() {
    return {
      slide: [
        {id:1,path:'static/images/1.jpg'},
        {id:2,path:'static/images/2.jpg'},
        {id:3,path:'static/images/3.jpg'},
      ],
      notNextTick: true,
      swiperOption: {
        autoplay: 3000,
        //          direction : 'vertical',
        grabCursor : true,
        setWrapperSize :true,
        autoHeight: true,
        pagination : '.swiper-pagination',
        paginationClickable :true,
        mousewheelControl : true,
        observeParents:true,
      }
    }
  }
}

```

列表页标签页是用swiper

调整Video.vue的swiper，可以查看swiper的deom

<https://surmon-china.github.io/vue-awesome-swiper/>

```

<!-- swiper -->
<swiper :options="swiperOption">
  <swiper-slide v-for="v in tags" :key="v.id">{{v.title}}</swiper-slide>
</swiper>

```

```
<!--导航条结束-->
```

```
export default {
    name: 'video',
    data() {
        return {
            tags:[
                {id:1,'title':'PHP'},
                {id:2,'title':'JS'},
                {id:3,'title':'PHP'},
                {id:4,'title':'JS'},
                {id:5,'title':'PHP'},
                {id:6,'title':'JS'},
            ],
            swiperOption: {
                pagination: '.swiper-pagination',
                slidesPerView: 3,
                spaceBetween: 30
            }
        }
    },
    components:{Bottom}
}
```

构建接口

- 标签接口

建立Api和Common控制器 php artisan make:controller Api/
ContentController php artisan make:controller Api/CommonController

CommonController的代码

```
abstract class CommonController extends Controller {
    protected function response( $data, $code = '200' ) {
        return [ 'data' => $data, 'code' => $code ];
    }
}
```

ContentController

```
class ContentController extends CommonController
```

```
{
    public function tags(){
        return $this->response(Tag::get());
    }
}
```

- }

建立路由

在web.php中

```
include __DIR__ . '/api.php';
```

在api.php中，访问/api/tags测试

- Route::group(
- ['prefix' => 'api', 'namespace' => 'Api'],
- function () {
- Route::match(['get'], '/tags', 'ContentController@tags');
-
- });

课程接口

```
Route::group(  
    [ 'prefix' => 'api', 'namespace' => 'Api' ],  
    function () {  
        Route::match( [ 'get' ], '/tags', 'ContentController@tags' );  
        Route::match( [ 'get' ], '/lessons/{tid}', 'ContentController@lessons' );  
  
    } );  
  
public function lessons( $tid ) {  
    if ( $tid ) {  
        $data = DB::table( 'lessons' )  
            ->select( 'lessons.*' )  
            ->join( 'lesson_tags', 'id', '=', 'lesson_id' )  
            ->where( [ 'tag_id' => $tid ] )  
            ->get();  
    } else {  
        $data = Lesson::get();  
    }  
  
    return $this->response( $data );  
}
```

推荐与热门接口

```
public function commendLessons($rows){  
    $data = Lesson::where(['iscommend'=>1])->limit($rows)->get();  
    return $this->response($data);  
}  
  
public function hotLessons($rows){  
    $data = Lesson::where(['ishot'=>1])->limit($rows)->get();  
    return $this->response($data);  
}  
Route::match( [ 'get' ], '/commendLessons/{rows}',  
    'ContentController@commendLessons' );  
Route::match( [ 'get' ], '/hotLessons/{rows}', 'ContentController@hotLessons' );
```

视频接口

```
public function videos($lid){  
    return $this->response(Video::where('lesson_id',$lid)->get());  
  
}  
Route::match( [ 'get' ], '/videos/{lid}', 'ContentController@videos' );
```

安装axios调用接口数据

在Npm官网搜索vue-axios

或者直接打开网址

<https://www.npmjs.com/search?q=vue-axios>

<https://github.com/imcvampire/vue-axios>

安装

```
cd webapp  
cnpm install --save axios vue-axios
```

在main.js

```
import axios from 'axios'  
import VueAxios from 'vue-axios'  
Vue.use(VueAxios, axios)
```

解决ajax跨域访问

默认情况下前台发送Ajax是允许跨域请求的。我们可以在后台进行相关设置然后允许前台跨域请求。

允许单个域名访问

```
header('Access-Control-Allow-Origin:http://www.houdunwang.com');
```

允许多个域名

```
$origin = isset($_SERVER['HTTP_ORIGIN'])? $_SERVER['HTTP_ORIGIN'] : '';
```

```
$allow_origin = array(  
    'http://www.houdunren.com',  
    'http://www.houdunwang.com'  
,);
```

```
if(in_array($origin, $allow_origin)){  
    header('Access-Control-Allow-Origin:'.$origin);  
}
```

允许所有域名请求，放入到router/api.php中

```
header('Access-Control-Allow-Origin:*');
```

推荐热门课程接口调用

```
data() {
```

```

        return {
            slide: [
                {id:1,path:'static/images/1.jpg'},
                {id:2,path:'static/images/2.jpg'},
                {id:3,path:'static/images/3.jpg'},
            ],
            commend:[],
            hot:[],
            notNextTick: true,
            swiperOption: {
                autoplay: 3000,
//                direction : 'vertical',
                grabCursor : true,
                setWrapperSize :true,
                autoHeight: true,
                pagination : '.swiper-pagination',
                paginationClickable :true,
                mousewheelControl : true,
                observeParents:true,
            }
        }
    },
    mounted(){
        this.axios.get('http://test.com/api/commendLessons/4').then((response)=>{
            this.commend = response.data.data;
        });
        this.axios.get('http://test.com/api/hotLessons/4').then((response)=>{
            this.hot = response.data.data;
        });
    },
<router-link :to="{name:'Page',params:{lid:v.id}}" v-for="v in commend" :key="v.id">
    
    <i class="iconfont icon-bofang"></i>
    <span class="time">22:56</span>
    <span class="title">{{v.title}}</span>
</router-link>

<router-link :to="{name:'Page',params:{lid:v.id}}" v-for="v in hot" :key="v.id">
    
</router-link>

```

路由

```

export default new Router({
  routes: [
    {
      path: '/',
      name: 'Home',
      component: Home
    },
    {
      path: '/video',
      name: 'Video',
      component: Video
    },
    {
      path: '/Page/:lid',
      name: 'Page',
      component: Page
    }
  ]
})

```

视频Page的接口调用

```

<li v-for="v in videos"><a @click.prevent="play(v)">{{v.title}}</a></li>

```

```

export default {
  name: 'Page',
  data() {
    return {
      videos:[],
      current:{}
    }
  },
  mounted(){
    let lid = this.$route.params.lid;
    this.axios.get('http://test.com/api/videos/' + lid).then((response)=>{
      if(response.status==200 && response.data.code==200){
        this.videos = response.data.data;
        this.current = this.videos[0];
      }else{
        alert('视频获取失败， 稍后再试');
      }
    })
  },
  methods:{
    play(v){
      this.current = v;
    }
  }
}

```

```
}

默认第一个视频
<video :src="current.path" controls="controls"></video>

返回按钮处理
<a href="" class="iconfont back" @click.prevent="back()">返回</a>

methods:{
  play(v){
    this.current = v;
  },
  back(){
    this.$router.back();
  }
}
```

获取标签

```
<!--导航条-->
<!-- swiper -->
<swiper :options="swiperOption">
  <swiper-slide v-for="v in tags" :key="v.id">
    <router-link :to="{name:'Video',params:{tid:v.id}}>
      {{v.name}}
    </router-link>
  </swiper-slide>
</swiper>
<!--导航条结束-->

<!--视频列表-->
<ul id="videolist">
  <li v-for="v in lessons">
    <router-link :to="{name:'Page',params:{lid:v.id}}" class="pic">
      
      <span>08:26</span>
      <i class="iconfont icon-bofang"></i>
    </router-link>
    <a href="" class="title">{{v.title}}</a>
  </li>
</ul>
<!--视频列表结束-->
export default {
  name: 'video',
  data() {
    return {
      tags: [],
      lessons: [],
      swiperOption: {
```

```
        pagination: '.swiper-pagination',
        slidesPerView: 3,
        spaceBetween: 30
    }
}
},
mounted() {
    this.loadData();
},
watch:{
    '$route'(to,from){
        this.loadData();
    }
},
methods:{
    loadData(){
        this.axios.get('http://test.com/api/tags').then((response) => {
            this.tags = response.data.data;

        })
        let tid = this.$route.params.tid;
        tid = tid ? tid : 0;
        this.axios.get('http://test.com/api/lessons/' + tid).then((response) => {
            this.lessons = response.data.data;

        })
    }
},
}
```

过渡动画效果

百度搜搜animate.css,安装animate.css

在App.vue设置

```
<transition enter-active-class="animated tada">
    <router-view></router-view>
</transition>
```

编译

把所有代码上传到服务器，然后设置好地址，
cnpm run build

让子域名绑定到dist目录

打包APP

注册登陆

<http://www.apicloud.com/signin>

1. 左侧创建应用，
2. 选择web

3. 输入你上线项目的网址
4. 选择右侧的云编译
5. 选择Android, 然后选择正式版, 点击云编译
6. 正式版要求上传证书,
7. 选择右上角的一键上传证书
8. 密码填写就可以了
9. 找到左侧的端设置然后上传图片
9. 再次云编译就可以上传证书了
10. 等一会就编译好了, 就可以安装了

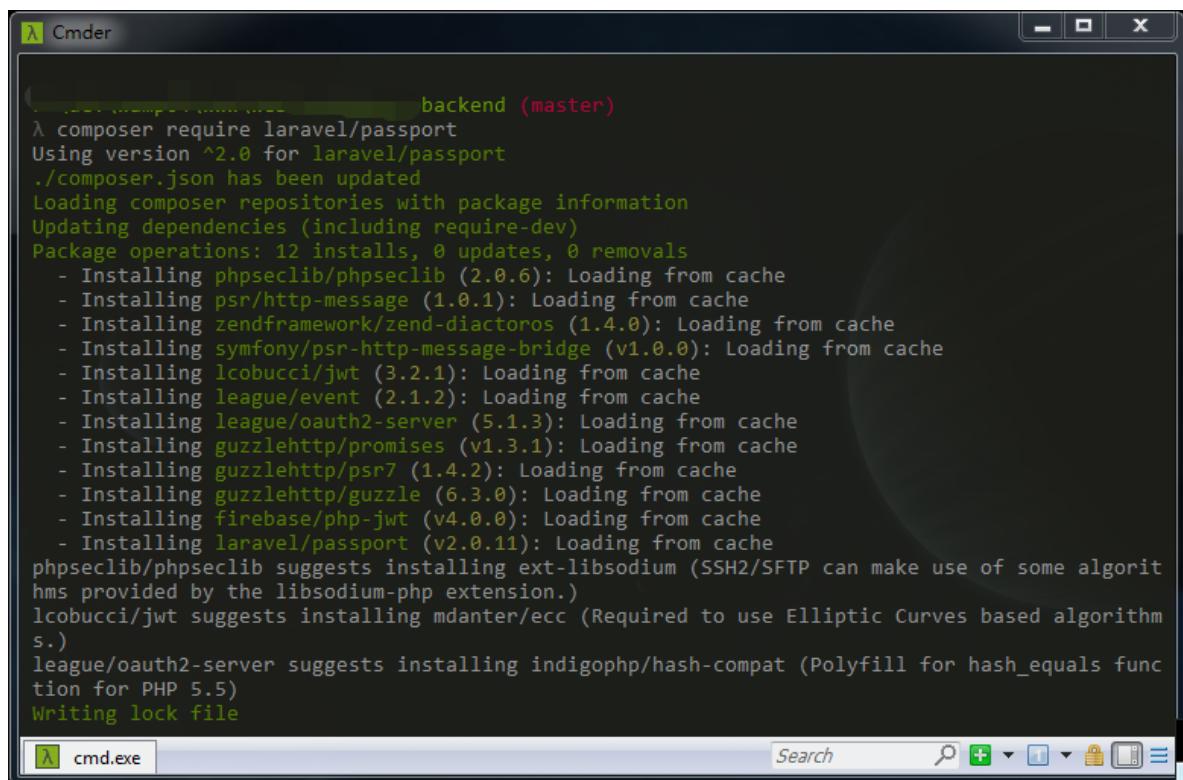
安装并使用 Laravel Passport 来请求授权令牌 (OAuth2.0认证方式)

一、安装 Passport

使用 Composer 依赖包管理器安装 Passport

composer require laravel/passport

1 composer require laravel/passport



```
λ composer require laravel/passport
Using version ^2.0 for laravel/passport
./composer.json has been updated
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 12 installs, 0 updates, 0 removals
- Installing phpseclib/phpseclib (2.0.6): Loading from cache
- Installing psr/http-message (1.0.1): Loading from cache
- Installing zendframework/zend-diactoros (1.4.0): Loading from cache
- Installing symfony/psr-http-message-bridge (v1.0.0): Loading from cache
- Installing lcobucci/jwt (3.2.1): Loading from cache
- Installing league/event (2.1.2): Loading from cache
- Installing league/oauth2-server (5.1.3): Loading from cache
- Installing guzzlehttp/promises (v1.3.1): Loading from cache
- Installing guzzlehttp/psr7 (1.4.2): Loading from cache
- Installing guzzlehttp/guzzle (6.3.0): Loading from cache
- Installing firebase/php-jwt (v4.0.0): Loading from cache
- Installing laravel/passport (v2.0.11): Loading from cache
phpseclib/phpseclib suggests installing ext-libodium (SSH2/SFTP can make use of some algorit
hms provided by the libsodium-php extension.)
lcobucci/jwt suggests installing mdanter/ecc (Required to use Elliptic Curves based algorithm
s.)
league/oauth2-server suggests installing indigophp/hash-compat (Polyfill for hash_equals func
tion for PHP 5.5)
Writing lock file
```

接下来将

Laravel\Passport\PassportServiceProvider::class,

1 Laravel\Passport\PassportServiceProvider::class,

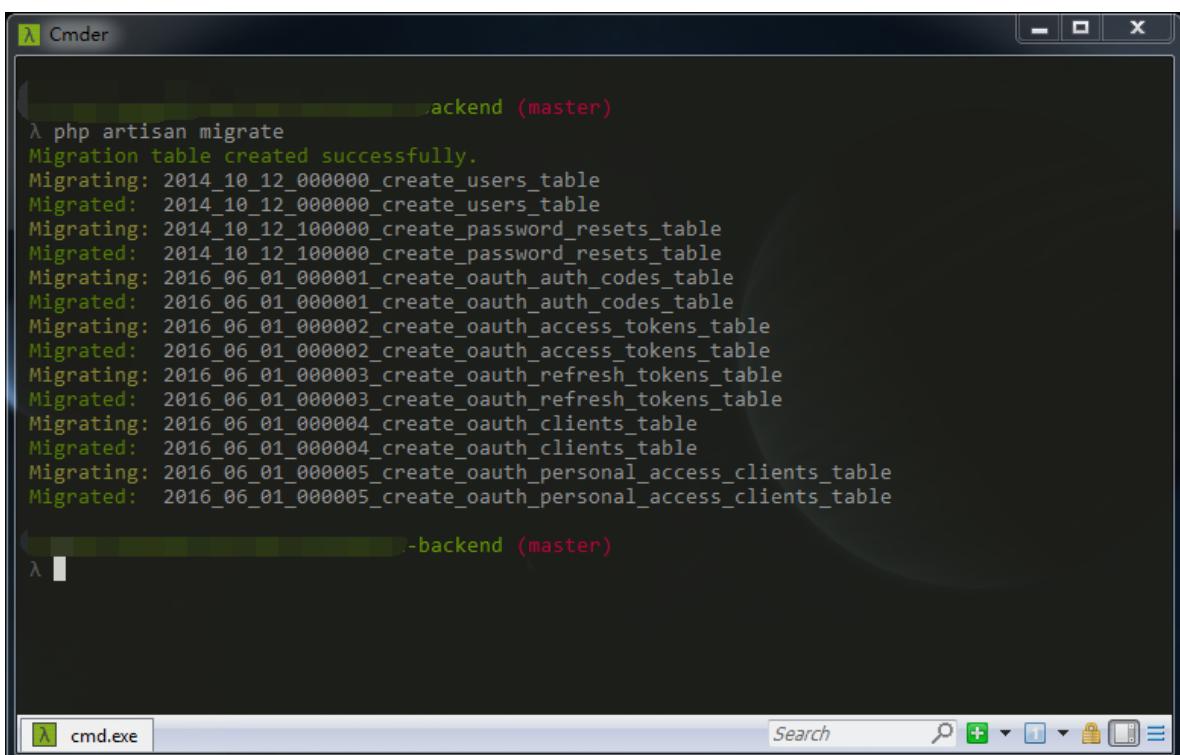
这句代码添加到 config/app.php 的 providers 数组,

```
App\Providers\AppServiceProvider::class,
App\Providers\AuthServiceProvider::class,
// App\Providers\BroadcastServiceProvider::class,
App\Providers\EventServiceProvider::class,
App\Providers\RouteServiceProvider::class,
  
    // Passport Provider
    Laravel\Passport\PassportServiceProvider::class,  
],  
/*  
 * Class Aliases  
 */
```

然后运行

```
php artisan migrate  
1 php artisan migrate
```

来运行 **Passport** 自带的数据库迁移文件（用于自动创建 **Passport** 必需的客户端数据表和令牌数据表），



```
λ php artisan migrate
Migration table created successfully.
Migrating: 2014_10_12_00000_create_users_table
Migrated: 2014_10_12_00000_create_users_table
Migrating: 2014_10_12_100000_create_password_resets_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrating: 2016_06_01_000001_create_oauth_auth_codes_table
Migrated: 2016_06_01_000001_create_oauth_auth_codes_table
Migrating: 2016_06_01_000002_create_oauth_access_tokens_table
Migrated: 2016_06_01_000002_create_oauth_access_tokens_table
Migrating: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrated: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrating: 2016_06_01_000004_create_oauth_clients_table
Migrated: 2016_06_01_000004_create_oauth_clients_table
Migrating: 2016_06_01_000005_create_oauth_personal_access_clients_table
Migrated: 2016_06_01_000005_create_oauth_personal_access_clients_table
```

然后运行

```
php artisan passport:install  
1 php artisan passport:install
```

来创建生成安全访问令牌时用到的加密密钥及私人访问和密码访问客户端。

```
Migrating: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrated: 2016_06_01_000003_create_oauth_refresh_tokens_table
Migrating: 2016_06_01_000004_create_oauth_clients_table
Migrated: 2016_06_01_000004_create_oauth_clients_table
Migrating: 2016_06_01_000005_create_oauth_personal_access_clients_table
Migrated: 2016_06_01_000005_create_oauth_personal_access_clients_table

λ php artisan passport:install
Encryption keys generated successfully.
Personal access client created successfully.
Client ID: 1
Client Secret: MAjKzqZoKAXilM4Arh7kM3dZdxjgqhtVQokSHhIA
Password grant client created successfully.
Client ID: 2
Client Secret: o6N7JzK2y3pbJiQmDcHaDhrpvPHBktnAlmmfEnS7
```

这是供私人访问的客户端
这是供密码访问的客户端

接下来再将

Laravel\Passport\HasApiTokens

1 Laravel\Passport\HasApiTokens

Trait 添加到 App\User 模型中，这个 Trait 会给这个模型提供一些辅助函数，用于检查已认证用户的令牌和使用作用于。

```
namespace App;

use Illuminate\Notifications\Notifiable;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravel\Passport\HasApiTokens;

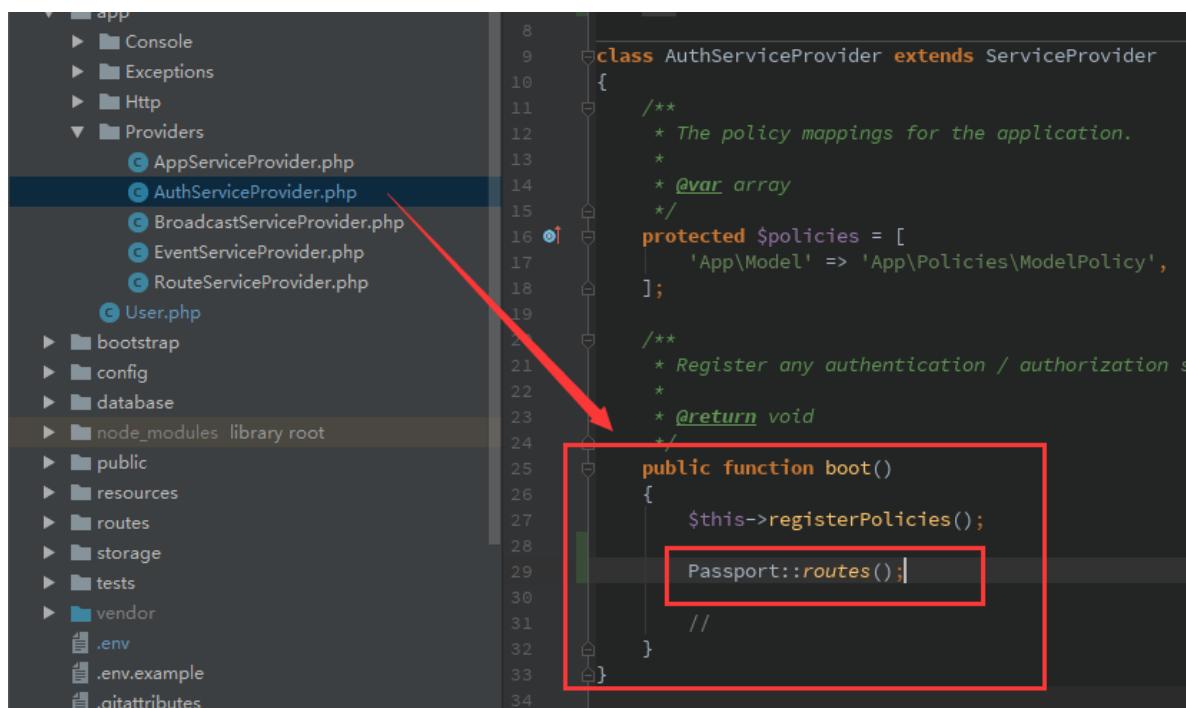
class User extends Authenticatable
{
    use Notifiable,HasApiTokens;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
```

然后在 AuthServiceProvider 的 boot 方法中添加

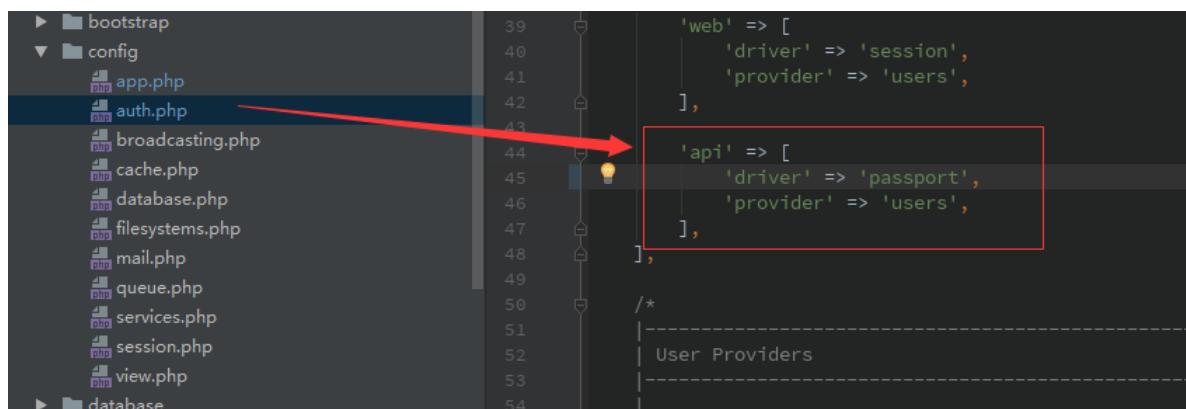
```
Passport::routes();  
1 Passport::routes();
```

来注册 **Passport** 在访问令牌、客户端、私人访问令牌的发放和吊销过程中一些必要路由。



```
class AuthServiceProvider extends ServiceProvider  
{  
    /**  
     * The policy mappings for the application.  
     *  
     * @var array  
     */  
    protected $policies = [  
        'App\Model' => 'App\Policies\ModelPolicy',  
    ];  
  
    /**  
     * Register any authentication / authorization services.  
     *  
     * @return void  
     */  
    public function boot()  
    {  
        $this->registerPolicies();  
        Passport::routes();  
    }  
}
```

最后修改下配置文件 config/auth.php 中的 api driver 改为 **passport**, 此项操作会将 API 请求时使用 **Passport** 的 **TokenGuard** 来处理授权保护。



```
'web' => [  
    'driver' => 'session',  
    'provider' => 'users',  
],  
  
'api' => [  
    'driver' => 'passport',  
    'provider' => 'users',  
],  
/*  
|-----  
| User Providers  
|-----
```

全部配置好后，我们来请求令牌，但是发现我们并没有创建任何用户，于是我们需要用到 **seed** 来预设填充数据。

运行

```
php artisan make:seeder UsersTableSeeder  
1 php artisan make:seeder UsersTableSeeder
```

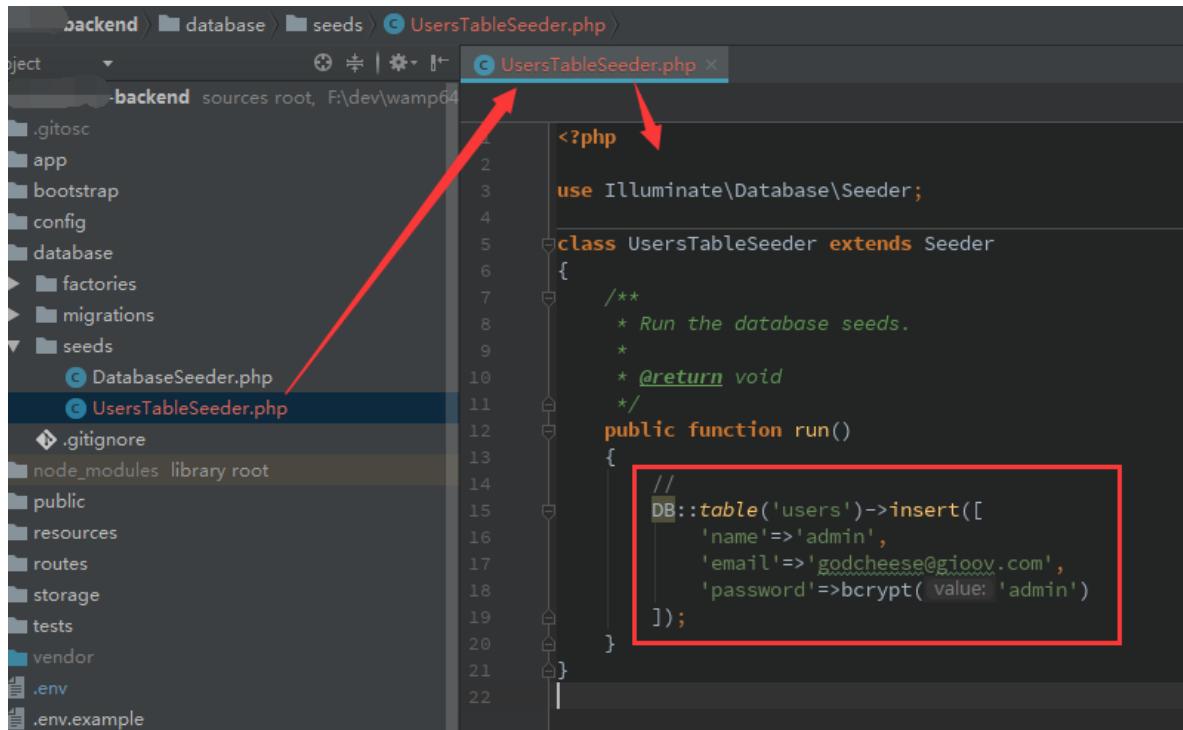
创建一个 Seeder

然后打开 database\seeds 文件夹里已经新建了一个叫 **UserTableSeeder.php** 的文件，打开它然后在 run 方法内输入以下代码：

```
DB::table('users')->insert([
    'name'=>'admin',
    'email'=>'godcheese@gioov.com',
    'password'=>bcrypt('admin')
]);
```

```
DB::table('users')->insert([
    'name'=>'admin',
    'email'=>'godcheese@gioov.com',
    'password'=>bcrypt('admin')
]);
```

如图所示，

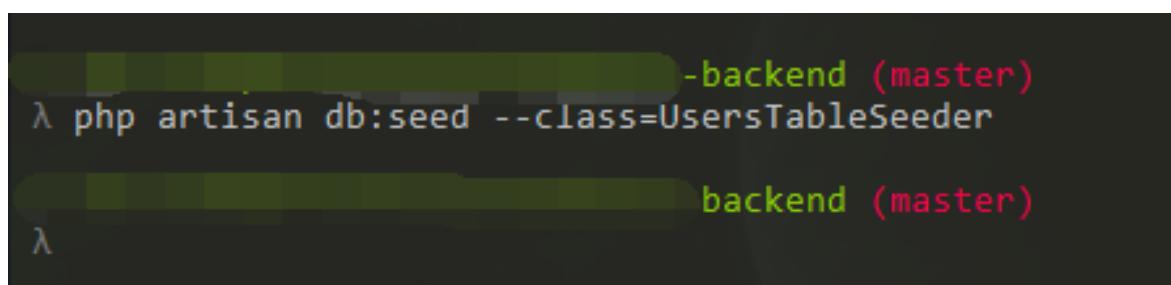


```
<?php
use Illuminate\Database\Seeder;

class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     *
     * @return void
     */
    public function run()
    {
        // DB::table('users')->insert([
        //     'name'=>'admin',
        //     'email'=>'godcheese@gioov.com',
        //     'password'=>brypt( value: 'admin')
        //]);
    }
}
```

然后运行

```
php artisan db:seed --class=UsersTableSeeder
1 php artisan db:seed --class=UsersTableSeeder
来往 user 表中填充刚刚添加好的 seed 数据。
```



```
-backend (master)
λ php artisan db:seed --class=UsersTableSeeder
backend (master)
λ
```

接下来我们来写前端 JavaScript 请求获取用户访问令牌的例子。

二、JavaScript 请求授权令牌

这是前端 UI，

The screenshot shows a login interface with the following elements:

- A text input field labeled "帐号" containing "admin". A red arrow points from this field to the text "标签id:username".
- A password input field labeled "密码" containing ".....". A red arrow points from this field to the text "标签id:password".
- A green button labeled "登录". A red arrow points from this button to the text "标签id:loginButton".

Below the form, there are two links: "忘了密码 ? 重置密码" and "还没账号 ? 注册帐号".

然后些下JavaScript请求代码如下：

```
$('#loginButton').click(function(event){  
event.preventDefault();  
$.ajax({  
type:'post',  
url:'http://wodedaxue-backend/oauth/token',  
dataType:'json',  
data:{  
'grant_type':'password',  
'client_id':'2',  
'client_secret':'o6N7JzK2y3pbJiQmDcHaDhrpvPHBktnAlmmfEnS7',  
'username':$('#username').val(),  
'password':$('#password').val(),  
'scope':''  
},  
success:function(data){  
console.log(data);  
alert(JSON.stringify(data));  
},  
error:function(err){  
console.log(err);  
alert('statusCode:'+err.status+'\n'+ 'statusText:' +err.statusText+'\n'+ 'description : \n'+JSON.stringify(err.responseJSON));  
}  
});  
});  
  
$('#loginButton').click(function(event){  
event.preventDefault();  
$.ajax({  
type:'post',  
url:'http://wodedaxue-backend/oauth/token',
```

```

dataType:'json',
data:{'grant_type':'password',
'client_id':'2',
'client_secret':'o6N7JzK2y3pbJiQmDcHaDhrpvPHBktnAlmmfEnS7',
'username':$('#username').val(),
'password':$('#password').val(),
'scope':''},
},
success:function(data){
console.log(data);
alert(JSON.stringify(data));
},
error:function(err){
console.log(err);
alert('statusCode:'+err.status+'\n'+statusText:'+err.statusText+'\n'+description
:\n'+JSON.stringify(err.responseJSON));
}
});
});
});

```

```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
backend resources views welcome.blade.php
Project
  - public
    - resources
      - assets
      - lang
    - views
      - welcome.blade.php
  - routes
  - storage
  - tests
  - vendor
    - .env
    - .env.example
    - .gitignore
    - artisan
    - composer.json
    - composer.lock
    - git-push.sh
    - LICENSE
    - npm-debug.log
    - package.json
    - phpuunit.xml
    - README.md
    - server.php
    - webpack.mix.js
  - External Libraries

```

```

<script type="text/javascript">
$(#LoginButton).click(function(event){
event.preventDefault();
$.ajax({
  type:'post',
  url:'http://localhost:8000/backend/oauth/token',
  dataType:'json',
  data:{
    'grant_type':'password',
    'client_id':'2',
    'client_secret':'o6N7JzK2y3pbJiQmDcHaDhrpvPHBktnAlmmfEnS7',
    'username':$('#username').val(),
    'password':$('#password').val(),
    'scope':''
  },
  success:function(data){
    console.log(data);
    alert(JSON.stringify(data));
  },
  error:function(err){
    console.log(err);
    alert('statusCode:'+err.status+'\n'+statusText:'+err.statusText+'\n'+description
:\n'+JSON.stringify(err.responseJSON));
  }
});
</script>
</body>

```

接下来再将以下代码写入 User 模型：

```

/**
 * 自定义用Passport授权登录：用户名+密码
 * @param $username
 * @return mixed
 */
public function findForPassport($username)
{

```

```

    return self::where('name', $username)->first();
}

/*
* 自定义用Passport授权登录：用户名+密码
* @param $username
* @return mixed
*/
public function findForPassport($username)
{
    return self::where('name', $username)->first();
}

```

The screenshot shows a code editor interface with the following details:

- File Path:** backend/app/User.php
- Code Editor:** The main pane displays the PHP code for the User class.
- Project Structure:** The left sidebar shows the project structure under "Project".
- User Class Definition:** The code defines a class "User" with protected properties \$fillable and \$hidden, and a public function findForPassport().

然后输入用户名 admin 和密码 admin , 点击登录按钮尝试登录：

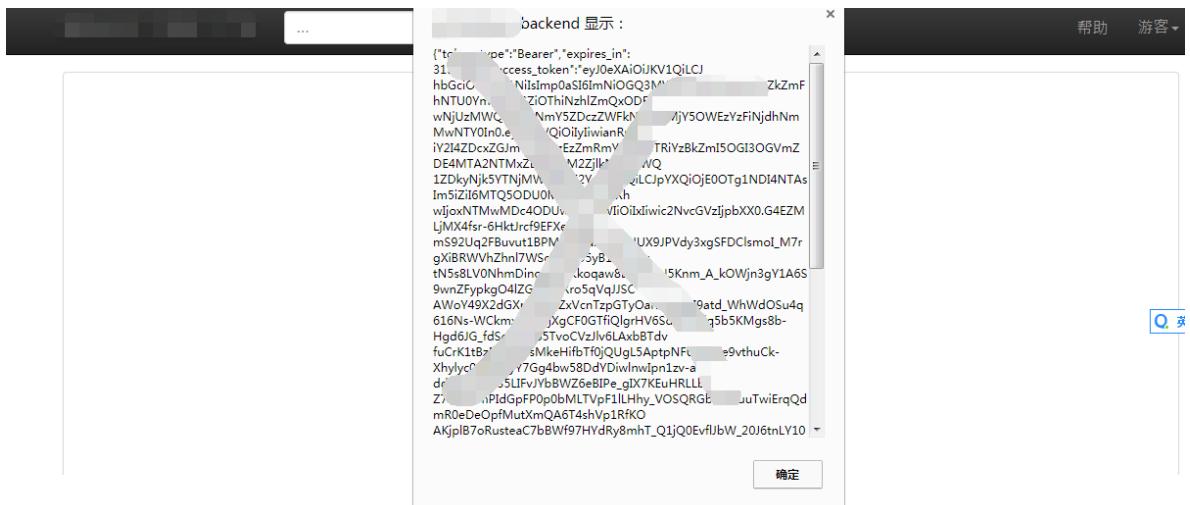
```

{
    token_type:"Bearer",
    expires_in:31536000,
    access_token:"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1Nlslmp0aSI6ImNiOG...
    MFF7PYRlhXqCkZhZNfoLV1IYTzRmhO2oV41XR9wD7Zd3Oxl-g",
    refresh_token:"sabtS2jYhqAJtVxe868OpTTgBeYgpWgfttn5h97drUgBVLN7vL...
    ukx7yGfGhn5h+zCouLmuh0sdjYU+dgtRweXU3jiA0Fa6/c6k="
}
{
    token_type:"Bearer",
    expires_in:31536000,
    access_token:"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1Nlslmp0aSI6ImNiOG...
    MFF7PYRlhXqCkZhZNfoLV1IYTzRmhO2oV41XR9wD7Zd3Oxl-g",
    refresh_token:"sabtS2jYhqAJtVxe868OpTTgBeYgpWgfttn5h97drUgBVLN7vL...
    ukx7yGfGhn5h+zCouLmuh0sdjYU+dgtRweXU3jiA0Fa6/c6k="
}

```

}

成功获取访问令牌 (access_token)。



然后在每次访问其它 API 时候都带上此参数。

Laravel Passport 更多参考资料：<https://laravel.com/docs/5.4/passport>

Laravel Passport 自定义获取授权用户的实现方案

最近有个项目使用了 Laravel Passport 来实现后端 API 的授权过程，整体用起来还是还顺利的，不过有个细节文档里面也并没有写，可以分享一下。

正常业务中，我们在通过请求中的 credentials 获取用户的时候，实际业务经常可能需要支持多个字段登入（比如同时去检查邮箱或者手机号码的匹配），或者需要检测用户是否被禁用之类的逻辑，可以统称为获取授权用户业务逻辑。

在 Laravel Passport 的源码中其实已经相应的去检查的对应 Guard 的 model 是否存在一个 findForPassport 方法：

```
# ./vendor/laravel/passport/src/Bridge/  
UserRepository@getUserEntityByUserCredentials
```

```
if (method_exists($model, 'findForPassport')) {  
    $user = (new $model)->findForPassport($username);  
} else {  
    $user = (new $model)->where('email', $username)->first();  
}
```

所以我们只需要给在 config/auth.php 配置的 Guard 对应的资源 model 添加一个 findForPassport 就方法可以，而不用再去通过中间件或者其他的方式实现：

```
# app\User.php
```

```
// 通过 phone 查找没有在禁用状态下的用户：  
public function findForPassport($username)  
{  
    return \App\User::normal()  
        ->where('phone', $username)
```

```
->first();  
}  
  
public function scopeNormal($query)  
{  
    return $query->where('status', self::STATUS_NORMAL);  
}  
Done!
```

使用Any授权管理系统模型

```
php artisan make:model Models\\DeptType  
php artisan any:repository DeptType  
php artisan make:request Admin\\DeptTypeRequest  
php artisan any:service Admin\\DeptType  
php artisan make:controller Admin\\DeptTypeController  
php artisan any:presenter Admin\\DeptType  
php artisan make:migration create_depttypes_table  
php artisan migrate
```