|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| 方圆ok1 | | | |
|  | **Web开发工具指导书** | |  |
|  | |
| **文档版本** | **01** |
| **发布日期** | **2019-01-20** |
| **freeair studio** | |

目录

[1 前言 1](#_Toc2610306)

[2 Windows设置 1](#_Toc2610307)

[2.1 右键菜单添加打开cmd 1](#_Toc2610308)

[3 文本编辑器 2](#_Toc2610309)

[3.1 安装Notepad++ 2](#_Toc2610310)

[3.2 安装VS Code 3](#_Toc2610311)

[3.2.1 转换为中文 3](#_Toc2610312)

[3.3 插件 4](#_Toc2610313)

[3.3.1 ESlint 4](#_Toc2610314)

[3.3.2 editorconfig 5](#_Toc2610315)

[3.3.3 Auto Close Tag 5](#_Toc2610316)

[3.3.4 Auto Rename Tag 5](#_Toc2610317)

[3.3.5 Prettier - Code formatter 6](#_Toc2610318)

[3.3.6 Better Align 6](#_Toc2610319)

[3.3.7 Better Comments 6](#_Toc2610320)

[3.3.8 Bookmarks 7](#_Toc2610321)

[3.3.9 Bracket Pair Colorizer 7](#_Toc2610322)

[3.3.10 gitignore 7](#_Toc2610323)

[3.3.11 Prettify JSON 7](#_Toc2610324)

[3.3.12 String Manipulation 7](#_Toc2610325)

[3.3.13 TODO Parser 7](#_Toc2610326)

[3.3.14 Vetur 8](#_Toc2610327)

[3.3.15 Phpfmt 8](#_Toc2610328)

[3.3.16 PHP DocBlocker 8](#_Toc2610329)

[3.4 安装Beyond Compare 9](#_Toc2610330)

[4 NPM 10](#_Toc2610331)

[4.1 概述 10](#_Toc2610332)

[4.2 安装Node.js 11](#_Toc2610333)

[4.3 修改npm模块路径和cache路径 12](#_Toc2610334)

[4.4 修改npm镜像 14](#_Toc2610335)

[4.5 安装cnpm（可选） 15](#_Toc2610336)

[4.6 删除模块 15](#_Toc2610337)

[4.7 Can't find Python 15](#_Toc2610338)

[4.8 工具集 16](#_Toc2610339)

[4.8.1 Svgo 16](#_Toc2610340)

[5 Webpack 17](#_Toc2610341)

[5.1 概述 17](#_Toc2610342)

[5.2 安装webpack 18](#_Toc2610343)

[5.3 安装加载器/Loader 19](#_Toc2610344)

[5.4 安装插件/Plugin 20](#_Toc2610345)

[6 Git 21](#_Toc2610346)

[6.1 用户信息 21](#_Toc2610347)

[6.2 创建本地仓库 21](#_Toc2610348)

[6.3 本地仓库添加文件 21](#_Toc2610349)

[6.4 本地仓库提交文件 21](#_Toc2610350)

[6.5 分支 21](#_Toc2610351)

[7 Github 23](#_Toc2610352)

[7.1 添加hosts 23](#_Toc2610353)

[7.2 SSH配置 23](#_Toc2610354)

[7.3 gitignore忽略文件 24](#_Toc2610355)

[7.3.1 忽略优先级 24](#_Toc2610356)

[7.3.2 忽略规则 25](#_Toc2610357)

[7.3.3 注意 26](#_Toc2610358)

[7.4 新建远程仓库 28](#_Toc2610359)

[7.5 本地仓库与远程仓库关联 28](#_Toc2610360)

[7.6 本地仓库推送至远程仓库 28](#_Toc2610361)

[7.7 克隆远程仓库 28](#_Toc2610362)

[7.8 从远程仓库更新本地仓库 29](#_Toc2610363)

[8 WAMP 30](#_Toc2610364)

[8.1 下载地址 30](#_Toc2610365)

[8.2 修改mysql的root密码 31](#_Toc2610366)

[8.3 修改phpadmin登录方式 31](#_Toc2610367)

[8.4 修改wampserver的www路径 31](#_Toc2610368)

[8.5 Apache Option 32](#_Toc2610369)

[8.6 Apache Rewrite模块 34](#_Toc2610370)

[8.6.1 基本点 34](#_Toc2610371)

[8.6.2 作用方法 34](#_Toc2610372)

[8.6.3 举例 35](#_Toc2610373)

[8.7 修改Apache根目录 38](#_Toc2610374)

[8.8 添加virtual host 39](#_Toc2610375)

[9 PHP Composer 40](#_Toc2610376)

[9.1 概述 40](#_Toc2610377)

[9.1.1 Composer 40](#_Toc2610378)

[9.2 安装 41](#_Toc2610379)

[9.2.1 解决与wamp的PATH环境变量冲突 41](#_Toc2610380)

[9.3 CN镜像 43](#_Toc2610381)

[9.3.1 全局更换镜像 43](#_Toc2610382)

[9.3.2 当前项目更换 43](#_Toc2610383)

[9.4 使用 44](#_Toc2610384)

[9.4.1 初始化项目 44](#_Toc2610385)

[9.4.2 添加依赖包 44](#_Toc2610386)

[9.4.3 安装依赖包 45](#_Toc2610387)

[9.4.4 代码引用依赖包 45](#_Toc2610388)

[9.4.5 更新项目依赖 46](#_Toc2610389)

[9.5 自动加载/autoload 47](#_Toc2610390)

[9.5.1 PSR-4 autoloading 47](#_Toc2610391)

[9.5.2 PSR-0 47](#_Toc2610392)

[9.5.3 Classmap 48](#_Toc2610393)

[9.5.4 Files 48](#_Toc2610394)

[9.6 PHP命名空间 49](#_Toc2610395)

[9.6.1 概述 49](#_Toc2610396)

[9.6.2 基础 49](#_Toc2610397)

[9.6.3 子空间 51](#_Toc2610398)

[9.6.4 公共空间 51](#_Toc2610399)

[9.6.5 名称术语 52](#_Toc2610400)

[9.6.6 别名和导入 52](#_Toc2610401)

[9.7 使用举例 54](#_Toc2610402)

[10 CodeIgniter和Vue框架 56](#_Toc2610403)

[10.1 目录结构 56](#_Toc2610404)

[10.2 CodeIgniter框架 56](#_Toc2610405)

[10.2.1 源文件路径 56](#_Toc2610406)

[10.2.2 移除url中的index.php 57](#_Toc2610407)

[10.2.3 使用php命名空间 58](#_Toc2610408)

[10.3 Vue框架Element admin 59](#_Toc2610409)

[10.3.1 源文件路径 59](#_Toc2610410)

[10.3.2 Element admin build设置 62](#_Toc2610411)

[10.3.3 单页面HTML5 History的后端设置 63](#_Toc2610412)

[10.4 CI项目引入iview admin 66](#_Toc2610413)

[10.4.1 CI目录添加iview-admin 66](#_Toc2610414)

[10.4.2 CI引入iviewadmin 68](#_Toc2610415)

[11 SCSS 71](#_Toc2610416)

[11.1 概述 71](#_Toc2610417)

[11.2 变量 71](#_Toc2610418)

[11.3 嵌套 72](#_Toc2610419)

[11.4 引入/@import 72](#_Toc2610420)

[11.5 混合/@mixin 74](#_Toc2610421)

[11.6 继承/@extend 74](#_Toc2610422)

[11.7 操作符 75](#_Toc2610423)

[11.8 CSS扩展 75](#_Toc2610424)

[11.8.1 引用符级选择器 75](#_Toc2610425)

[11.8.2 嵌套属性 77](#_Toc2610426)

[12 附录 79](#_Toc2610427)

[12.1 A 79](#_Toc2610428)

[12.1.1 79](#_Toc2610429)

# 前言

修编人员

|  |  |  |  |
| --- | --- | --- | --- |
| 编写人 | 桑宇 | 日期 | 2019-01-20 |
| 审核人 |  | 日期 |  |
| 批准人 |  | 日期 |  |

读者对象

本文档帮助您了解Web前端与后端开发用到的相关工具软件，框架应用和相关设置。

符号约定

在本文中可能出现下列标志，它们所代表的含义如下。

| 符号 | 说明 |
| --- | --- |
|  | 用于传递安全警示信息。 |
|  | 用于突出重要/关键信息、最佳实践和小窍门等。 |

命令行格式约定

在本文中可能出现下列命令行格式,它们所代表的含义如下。

| 格式 | 意义 |
| --- | --- |
| **粗体** | 命令行关键字（命令中保持不变、必须照输的部分）采用**加粗**字体表示。 |
| *斜体* | 命令行参数（命令中必须由实际值进行替代的部分）采用*斜体*表示。 |
| [ ] | 表示用“[ ]”括起来的部分在命令配置时是可选的。 |
| { x | y | ... } | 表示从两个或多个选项中选取一个。 |
| [ x | y | ... ] | 表示从两个或多个选项中选取一个或者不选。 |
| { x | y | ... } \* | 表示从两个或多个选项中选取多个，最少选取一个，最多选取所有选项。 |
| [ x | y | ... ] \* | 表示从两个或多个选项中选取多个或者不选。 |
| &<1-n> | 表示符号“&”前面的参数可以重复1～n次。 |
| # | 表示由“#”开始的行为注释行。 |

图形界面元素引用约定

在本文中可能出现下列图形界面元素，它们所代表的含义如下。

| 格式 | 意义 |
| --- | --- |
| “” | 带双引号“”的格式表示各类界面控件名称和数据表，如单击“确定”。 |
| > | 多级菜单用“>”隔开。如选择“文件>新建>文件夹”，表示选择“文件”菜单下的“新建”子菜单下的“文件夹”菜单项。 |

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 01 (**2019-01-20**)

第一次正式发布。

# Windows设置

## 右键菜单添加打开cmd

在HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\Directory\background\shell 下新建->项

名字: icmd

默认值: 在此处打开cmd命令行

(注：自定义默认值)

在HKEY\_LOCAL\_MACHINE\SOFTWARE\Classes\Directory\background\shell\icmd下新建->项

名字: command

默认值: C:\Windows\System32\cmd.exe

# 文本编辑器

## 安装Notepad++

## 安装VS Code

### 转换为中文

安装“Chinese”插件



重启VS Code

## 插件

### ESlint

ESLint 是一个ECMAScript/JavaScript 语法规则和代码风格的检查工具，先决条件：Node.js (>=4.x), npm version 2+

--------------

**webpack引入ESlint：**

首先，要使webpack支持eslint，就要要安装 eslint-loader。

在webpack.config.js 中添加如下代码：

{

test: /\.(js|vue)$/,

loader: 'eslint-loader',

enforce: 'pre',

include: [resolve('src'), resolve('test')],

options: {

formatter: require('eslint-friendly-formatter'),

emitWarning: !config.dev.showEslintErrorsInOverlay

}

}

其次，要想webpack具有 eslint 的能力，就要安装eslint。

最后，项目想要使用那些eslin规则，可以创建一个配置项文件 ‘.eslintrc.js’

--------------

VS Code中使用ESlint：

首先安装eslint插件。

VSCode 进行扩展设置，依次点击 文件 > 首选项 > 设置 打开 VSCode 配置文件,添加如下配置。

"files.autoSave":"off",

"eslint.validate": [

"javascript",

"javascriptreact",

"html",

{ "language": "vue", "autoFix": true }

],

"eslint.options": {

"plugins": ["html"]

}

### editorconfig

插件功能不用手工启动。

root = true

[\*]

charset = utf-8

indent\_style = space

indent\_size = 2

end\_of\_line = lf

insert\_final\_newline = true

trim\_trailing\_whitespace = true

# end\_of\_line 保存文件时，触发

# insert\_final\_newline 保存文件时，触发

# trim\_trailing\_whitespace 保存文件时，触发

[\*.md]

insert\_final\_newline = false

trim\_trailing\_whitespace = false

### Auto Close Tag

输入标签，自动闭合HTML标签。

缺少闭合标签的行，Alt+. 添加闭合标签。

### Auto Rename Tag

修改HTML标签时，自动修改匹配的标签

### Prettier - Code formatter

在格式化代码方面， Prettier 确实和 ESLint 有重叠，但两者侧重点不同：ESLint 主要工作就是检查代码质量并给出提示，它所能提供的格式化功能很有限；而 Prettier 在格式化代码方面具有更大优势。而 Prettier 被设计为易于与 ESLint 集成，所以你可以轻松在项目中使两者，而无需担心冲突。

对于二者集成，可以利用插件来完成；eslint-plugin-prettier 公开了一个 “recommended” 配置，将 plugin:prettier/recommended 添加到 extends 的子属性 plugin:vue/essential 之后，以默认设置在 ESLint 中启用对 Prettier 的支持。

extends: ['plugin:vue/recommended', "plugin:prettier/recommended", 'eslint:recommended'],

还需安装依赖库： eslint-plugin-prettier 和 eslint-config-prettier

### Better Align

对齐赋值符号和注释

Place your cursor at where you want your code to be aligned, and invoke the Align command via Command Palette or customized shortcut. Then the code will be automatically aligned

There's no built-in shortcut comes with the extension, you have to add shotcuts by yourself:

Open Command Palette and type open shortcuts to open keybinding settings

Add something similar like this:

{

"key": "ctrl+alt+A",

"command": "wwm.aligncode",

"when": "editorTextFocus && !editorReadonly"

}

### Better Comments

注释添加颜色

/\*\*

\* \* A

\* ! B

\* ? C

\* TODO: D

\* @param F

\*/

### Bookmarks

鼠标右键菜单操作。

### Bracket Pair Colorizer

自动用不同颜色高亮显示匹配的括号

### gitignore

gitignore文件语法

### Prettify JSON

格式化JSON

### String Manipulation

字符串转换处理（驼峰、大写开头、下划线等等）

使用：

选中字符串，命令面板输入string manipulation，选择转换的样式。

### TODO Parser

解析注释TODO

We support both single-line and multi-line comments. For example:

// TODO: this todo is valid

/\* TODO: this is also ok \*/

/\* It's a nice day today

\* Todo: multi-line TODOs are

\* supported too!

\*/

使用：

状态栏显示当前文件的TODO数目；

F1输入栏，输入：Parse TODOs

### Vetur

### Phpfmt

自动格式化php代码：phpfmt - PHP formatter

### PHP DocBlocker

生成函数的注释（/\*\*）

## 安装Beyond Compare

w4G-in5u3SH75RoB3VZIX8htiZgw4ELilwvPcHAIQWfwfXv5n0IHDp5hv1BM3+H1XygMtiE0-JBgacjE9tz33sIh542EmsGs1yg638UxVfmWqNLqu-Zw91XxNEiZF7DC7-iV1XbSfsgxI8Tvqr-ZMTxlGCJU+2YLveAc-YXs8ciRTtssts7leEbJ979H5v+G0sw-FwP9bjvE4GCJ8oj+jtlp7wFmpVdzovEhv5Vg3dMqhqTiQHKfmHjYbb0o5OUxq0jOWxg5NKim9dhCVF+avO6mDeRNcOYpl7BatIcd6tsiwdhHKRnyGshyVEjSgRCRY11IgyvdRPnbW8UOVULuTE

# NPM

## 概述

NPM（node package manager），通常称为node包管理器。顾名思义，它的主要功能就是管理node包，包括：安装、卸载、更新、查看、搜索、发布等。

npm的背后，是基于couchdb的一个数据库，详细记录了每个包的信息，包括作者、版本、依赖、授权信息等。它的一个很重要的作用就是：将开发者从繁琐的包管理工作（版本、依赖等）中解放出来，更加专注于功能的开发。

NPM是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种：

允许用户从NPM服务器下载别人编写的第三方包到本地使用。

允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。

允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

## 安装Node.js

## 修改npm模块路径和cache路径

将global模块和cache缓存路径修改到D盘目录下，手动创建文件加node\_cache和node\_global.



执行命令：

npm config set prefix "D:\nodejs\node\_global"

npm config set cache "D:\nodejs\node\_cache"



系统变量，新建"NODE\_PATH"，输入D:\nodejs\node\_global\node\_modules



用户变量，PAHT添加D:\nodejs\node\_global



## 修改npm镜像

使用NPM安装依赖时速度特别慢，在安装时可以手动指定从哪个镜像服务器获取资源，我们可以使用阿里巴巴在国内的镜像服务器，命令如下：

npm install -gd express --registry=http://registry.npm.taobao.org

只需要使用–registry参数指定镜像服务器地址，为了避免每次安装都需要--registry参数，可以使用如下命令进行永久设置：

npm config set registry <http://registry.npm.taobao.org>

## 安装cnpm（可选）

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

## 删除模块

【npm uninstall xxx】删除xxx模块；

【npm uninstall -g xxx】删除全局模块xxx；

## Can't find Python

npm --add-python-to-path='true' --debug install --global windows-build-tools

npm install --global --production windows-build-tools

## 工具集

### Svgo

为什么需要？

因为SVG文件，尤其从各种变假期导出的SVG，通常包含大量的无用信息，例如编辑器源信息，注释，因此元素，默认或者非最优值，以及其他一些不会影响渲染结果的可以移除或转换的内容。

能做什么？

SVGO基于插件模式构建，基本上所有的优化都是一个分离的插件。

如何使用？

首先是安装，连我都驾轻就熟了，如下：

$ [sudo] npm install -g svgo

单个文件使用举例：

$ svgo test.svg

或者:

$ svgo test.svg test.min.svg

文件夹举例：

$ svgo -f ../path/to/folder/with/svg/files

或者：

$ svgo -f ../path/to/folder/with/svg/files -o ../path/to/folder/with/svg/output

# Webpack

## 概述

Webpack 是一个前端资源加载/打包工具。它将根据模块的依赖关系进行静态分析，然后将这些模块按照指定的规则生成对应的静态资源。

WebPack可以看做是模块打包机：它做的事情是，分析你的项目结构，找到JavaScript模块以及其它的一些浏览器不能直接运行的拓展语言（Scss，TypeScript等），并将其打包为合适的格式以供浏览器使用。

在webpack看来一切都是模块，包括你的JavaScript代码，也包括CSS和fonts以及图片等等，只有通过合适的loaders，它们都可以被当做模块被处理。

Loaders和Plugins常常被弄混，但是他们其实是完全不同的东西，可以这么来说，loaders是在打包构建过程中用来处理源文件的（JSX，Scss，Less..），一次处理一个，Plugins并不直接操作单个文件，它直接对整个构建过程其作用。



Webpack 可以将多种静态资源 js、css、less 转换成一个静态文件，减少了页面的请求。

## 安装webpack

npm install webpack -g

npm install webpack-cli -g

## 安装加载器/Loader

CSS 样式Loader

npm install css-loader -g

npm install style-loader -g

图片和字体等Loader

npm install url-loader -g

npm install file-loader -g

## 安装插件/Plugin

extract-text-webpack-plugin

npm install extract-text-webpack-plugin -g

打包依赖插件

npm install webpack-merge -g

npm install html-webpack-plugin -g

# Git

## 用户信息

命令行输入

git config --global user.name "xx"

git config --global user.email [xx@163.com](mailto:xx@163.com)

git config --global user.name ""

git config --global user.email ""

## 创建本地仓库

切换到保存仓库目录

命令：git init

## 本地仓库添加文件

添加指定文件：git add filename

添加当前路径下所有文件：git add .

## 本地仓库提交文件

命令：git commit -m "备注"

## 分支

创建分支

git checkout -b 分支名

-b参数表示创建并切换，相当于以下两条命令

$ git branch dev

$ git checkout dev

合并分支

git merge分支名

删除分支

git branch -d 分支名

切换分支

git checkout分支名

# Github

## 添加hosts

151.101.185.194 github.global.ssl.fastly.net.ipaddress.com

192.30.253.112 github.com

## SSH配置

本地Git仓库和GitHub仓库之间的传输是通过SSH加密。在用户主目录下，看看有没有.ssh目录，如果有，再看看这个目录下有没有id\_rsa和id\_rsa.pub这两个文件。

ssh-keygen -t rsa -C [youremail@example.com](mailto:youremail@example.com)

id\_rsa是私钥，id\_rsa.pub是公钥

登陆GitHub，打开“Account settings”，“SSH Keys”页面

在Key文本框里粘贴id\_rsa.pub文件的内容

## gitignore忽略文件

### 忽略优先级

在 .gitingore文件中，每一行指定一个忽略规则，Git检查忽略规则的时候有多个来源，它的优先级如下（由高到低）：

从命令行中读取可用的忽略规则

当前目录定义的规则

父级目录定义的规则，依次递推

$GIT\_DIR/info/exclude 文件中定义的规则

core.excludesfile中定义的全局规则配置运维总结

### 忽略规则

空格不匹配任意文件，可作为分隔符，可用反斜杠转义

以“＃”开头的行都会被Git忽略。即#开头的文件标识注释，可以使用反斜杠进行转义。

可以使用标准的glob模式匹配。所谓的glob模式是指shell所使用的简化了的正则表达式。

以斜杠"/"开头表示目录；

"/"结束的模式只匹配文件夹以及在该文件夹路径下的内容，但是不匹配该文件；

"/"开始的模式匹配项目跟目录；

如果一个模式不包含斜杠，则它匹配相对于当前 .gitignore文件路径的内容，如果该模式不在 .gitignore文件中，则相对于项目根目录。

以星号"\*"通配多个字符，即匹配多个任意字符；使用两个星号"\*\*" 表示匹配任意中间目录，比如`a/\*\*/z`可以匹配 a/z, a/b/z 或 a/b/c/z等。

以问号"?"通配单个字符，即匹配一个任意字符；

以方括号"[]"包含单个字符的匹配列表，即匹配任何一个列在方括号中的字符。比如[abc]表示要么匹配一个a，要么匹配一个b，要么匹配一个c；如果在方括号中使用短划线分隔两个字符，表示所有在这两个字符范围内的都可以匹配。比如[0-9]表示匹配所有0到9的数字，[a-z]表示匹配任意的小写字母）。

以叹号"!"表示不忽略(跟踪)匹配到的文件或目录，即要忽略指定模式以外的文件或目录，可以在模式前加上惊叹号（!）取反。需要特别注意的是：如果文件的父目录已经被前面的规则排除掉了，那么对这个文件用"!"规则是不起作用的。也就是说"!"开头的模式表示否定，该文件将会再次被包含，如果排除了该文件的父级目录，则使用"!"也不会再次被包含。可以使用反斜杠进行转义。





### 注意

如果你不慎在创建.gitignore文件之前就push了项目，那么即使你在.gitignore文件中写入新的过滤规则，这些规则也不会起作用，Git仍然会对所有文件进行版本管理。简单来说出现这种问题的原因就是Git已经开始管理这些文件了，所以你无法再通过过滤规则过滤它们。所以大家一定要养成在项目开始就创建.gitignore文件的习惯，否则push后，处理起来会非常麻烦。

## 新建远程仓库

## 本地仓库与远程仓库关联

本地仓路径下，执行命令：git remote add origin [git@github.com:xxx/yyy.git](mailto:git@github.com:xxx/yyy.git)

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这个名字一看就知道是远程库。

[xxx/yyy.git](mailto:git@github.com:xxx/yyy.git)在github页面获取。

## 本地仓库推送至远程仓库

命令：git push -u origin master

用git push命令，实际上是把当前分支master推送到远程。

-u参数，Git不但会把本地的master分支内容推送的远程新的master分支，还会把本地的master分支和远程的master分支关联起来，在以后的推送或者拉取时就可以简化命令：git push origin master

## 克隆远程仓库

命令：git clone [git@github.com:xxx/yyy.git](mailto:git@github.com:xxx/yyy.git)

## 从远程仓库更新本地仓库

Git pull

//查询当前远程的版本

$ git remote -v

//直接拉取并合并最新代码

$ git pull origin master [示例1：拉取远端origin/master分支并合并到当前分支]

$ git pull origin dev [示例2：拉取远端origin/dev分支并合并到当前分支]

直接合并，无法提前处理冲突

git fetch

//查询当前远程的版本

$ git remote -v

//获取最新代码到本地(本地当前分支为[branch]，获取的远端的分支为[origin/branch])

$ git fetch origin master [示例1：获取远端的origin/master分支]

$ git fetch origin dev [示例2：获取远端的origin/dev分支]

//查看版本差异

$ git log -p master..origin/master [示例1：查看本地master与远端origin/master的版本差异]

$ git log -p dev..origin/dev [示例2：查看本地dev与远端origin/dev的版本差异]

//合并最新代码到本地分支

$ git merge origin/master [示例1：合并远端分支origin/master到当前分支]

$ git merge origin/dev [示例2：合并远端分支origin/dev到当前分支]

# WAMP

## 下载地址

<https://sourceforge.net/projects/wampserver/files/>

If you have a 64-bit Windows, you must install both 32 and 64bit versions of each VisualC++ package, even if you do not use Wampserver 64 bit.



## 修改mysql的root密码

use mysql;

update user set authentication\_string=PASSWORD('newpw123') where user='root';

flush privileges;

## 修改phpadmin登录方式

文件：D:\wamp64\apps\phpmyadmin4.5.2config.inc.php



## 修改wampserver的www路径

文件：wampserver的安装目录，/script/config.inc.php

$wwwDir = $c\_installDir.'/www';,改为指定目录$wwwDir = 'D:/www';

## Apache Option

Options指令可以在Apache服务器核心配置(server config)、虚拟主机配置(virtual host)、特定目录配置(directory)以及.htaccess文件中使用。Options指令的主要作用是控制特定目录将启用哪些服务器特性。

1. Indexes，控制显示目录列表

开启：如果文件根目录里有 index.html，浏览器就会显示 index.html的内容，如果没有 index.html，浏览器就会显示文件根目录的目录列表，目录列表包括文件根目录下的文件和子目录。

在Indexes前面加上 – 符号，为关闭。

1. FollowSymLinks，服务器允许在此目录中使用符号连接(如果该配置选项位于配置段中，将会被忽略)
2. MultiViews，允许使用mod\_negotiation模块提供内容协商的”多重视图”。简而言之，如果客户端请求的路径可能对应多种类型的文件，那么服务器将根据客户端请求的具体情况自动选择一个最匹配客户端要求的文件。例如，在服务器站点的file文件夹下中存在名为hello.jpg和hello.html的两个文件，此时用户输入Http://localhost/file/hello，如果在file文件夹下并没有hello子目录，那么服务器将会尝试在file文件夹下查找形如hello.\*的文件，然后根据用户请求的具体情况返回最匹配要求的hello.jpg或者hello.html
3. SymLinksIfOwnerMatch，服务器仅在符号连接与目标文件或目录的所有者具有相同的用户ID时才使用它。简而言之，只有当符号连接和符号连接指向的目标文件或目录的所有者是同一用户时，才会使用符号连接(如果该配置选项位于配置段中，将会被忽略)
4. ExecCGI，允许使用mod\_cgi模块执行CGI脚本
5. Includes，允许使用mod\_include模块提供的服务器端包含功能
6. IncludesNOEXEC，允许服务器端包含，但禁用”#exec cmd”和”#exec cgi”。但仍可以从ScriptAlias目录使用”#include virtual”虚拟CGI脚本
7. All，表示除MultiViews之外的所有特性（这也是Options指令的默认设置）
8. None，不启用任何的服务器特性

Apache允许在一个目录配置中设置多个Options指令。不过，一般来说，如果一个目录被多次设置了Options，则指定特性数量最多的一个Options指令会被完全接受(其它的被忽略)，而各个Options指令之间并不会合并。但是如果我们在可选配置项前加上了符号”+”或”-“，那么表示该可选项将会被合并。所有前面加有”+”号的可选项将强制覆盖当前的可选项设置，而所有前面有”-“号的可选项将强制从当前可选项设置中去除。你可以参考下面的例子：

#示例1

<Directory /web/file>

Options Indexes FollowSymLinks

</Directory>

<Directory /web/file/image>

Options Includes

</Directory>

#目录/web/file/image只会被设置Includes特性

#示例2

<Directory /web/file>

Options Indexes FollowSymLinks

</Directory>

<Directory /web/file/image>

Options +Includes -Indexes

</Directory>

#目录/web/file/image将会被设置Includes、FollowSymLinks两种特性

## Apache Rewrite模块

### 基本点

1. 重写规则：RewriteRule Pattern Substitution [flags]
2. ！是个合法的模板前缀，表示“非”的意思，这对描述“不满足某种匹配条件”的情况非常方便，或用作最后一条默认规则。当使用！时，不能在模板中有分组的通配符，也不能做后向引用
3. 当匹配成功后，Substitution会被用来替换相应的匹配，它除了可以是普通的字符串以外，还可以包括：

1. $N,引用RewriteRule模板中匹配的相关字串，N表示序号,N=0..9

2. %N,引用最后一个RewriteCond模板中匹配的数据，N表示序号

3. %{VARNAME},服务器变量

4. ${mapname:key|default},映射函数调用

1. Flags是可选参数，当有多个标志同时出现时，彼此间以逗号分隔。

1. 'redirect|R [=code]' (强制重定向)

给当前的URI增加前缀http://thishost[:thisport]/， 从而生成一个新的URL，强制生成一个外部重定向(external redirection，指生的URL发送到客户端，由客户端再次以新的URL发出请求，虽然新URL仍指向当前的服务器). 如果没有指定的code值，则HTTP应答以状态值302 (MOVED TEMPORARILY)，如果想使用300-400（不含400）间的其它值可以通过在code的位置以相应的数字指定，也可以用标志名指定： temp (默认值), permanent, seeother.

注意，当使用这个标志时，要确实substitution是个合法的URL，这个标志只是在URL前增加http://thishost[:thisport]/前缀而已，重写操作会继续进行。如果要立即将新URL重定向，用L标志来中重写流程。

5. 'last|L' (最后一条规则)

中止重写流程，不再对当前URL施加更多的重写规则。这相当于perl的last命令或C的break命令。

10. 'nocase|NC' (模板不区分大小写)

这个标志会使得模板匹配当前URL时忽略大小写的差别。

### 作用方法

1) 可以使用在Apache主配置文件httpd.conf中

2) 可以使用在httpd.conf里定义的虚拟主机配置中

3) 可以使用在基本目录的跨越配置文件.htaccess中，需要确定主配置文件中对你的网站所在的目录定义了下面的内容，否则你的.htaccess不会工作。

Options Indexes FollowSymLinks

AllowOverride all

### 举例

对该目录的任何请求都重定向到另一个文件

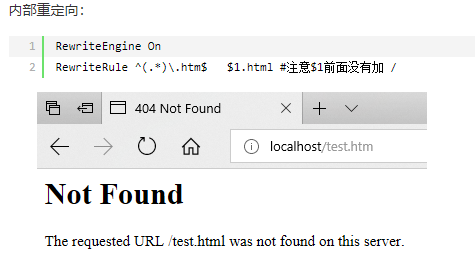
rewriteengine on

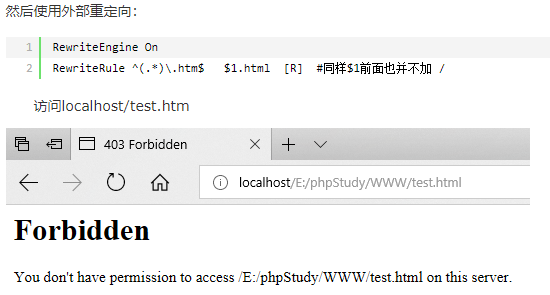
rewritebase /

rewriterule ^(.\*)$ index.html.en [R=301]

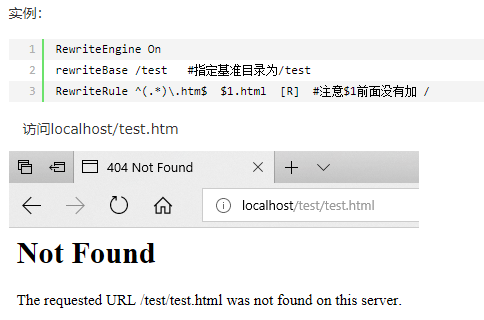
RewriteBase可以定义基准路径，上面将当前目录设置为基准目录。

RewriteBase用于设置目录级重写的基准URL，即所有的重定向都是基于这个URL。内部重定向可能看不出效果，但是在外部重定向（使用R flag后），如果不手动指定 / 为根目录，那么就会去整个磁盘路径查找。







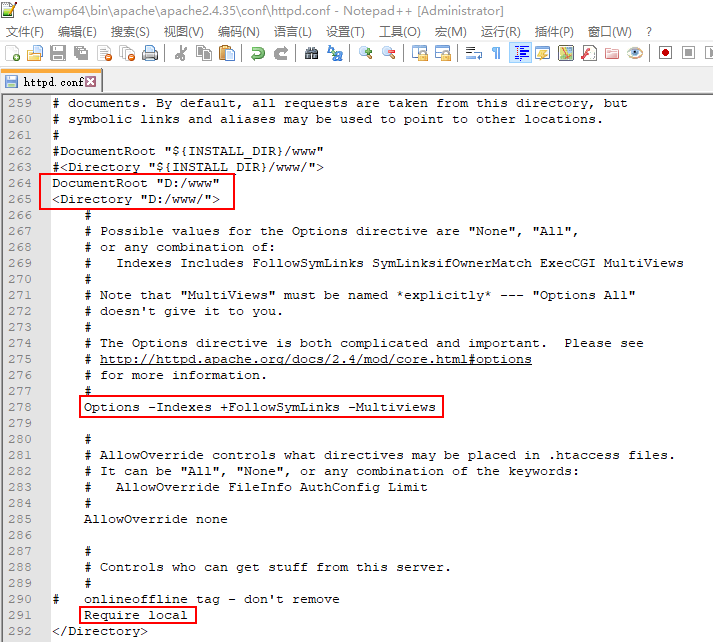


rewritebase / 的作用是如果在下面的rewriterule定义中被重写后的部分(此处为文件名index.html.en)前面没有/，则是相对目录。

## 修改Apache根目录

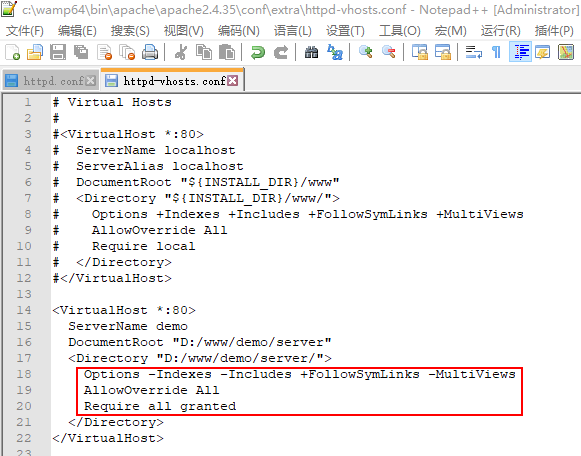
文件：http.conf

查找“DocumentRoot”, 修改后面的路径，改为指定项目目录。



## 添加virtual host

不同虚拟主机对应不同网站实例，方便切换。开启Option- FollowSymLinks，开启重写AllowOverride All，允许所有来源访问Require all granted



# 

# PHP Composer

## 概述

### Composer

Composer服务于 PHP 生态系统，包含了两个部分：Composer 和 Packagist。其利用了 PSR-0 到PSR-4 以及 PHP5.3 的命名空间。

为项目自动安装所依赖的开发包。Composer 包含了一个依赖解析器，用来处理开发包之间复杂的依赖关系；另外，它还包含了下载器、安装器等有趣的东西。

在 composer.json 文件中声明当前项目所依赖的开发包，然后运行 composer install 就行了。composer.json 文件定义了当前项目所依赖的开发包和 composer 的配置信息。

php代码规范PSR简介

Framework Interoperability Group（框架可互用性小组），简称 FIG，FIG 提出了 PSR-0 到 PSR-4 五套 PHP 非官方规范：

PSR-0 (Autoloading Standard) 自动加载标准

PSR-1 (Basic Coding Standard) 基础编码标准

PSR-2 (Coding Style Guide) 编码风格向导

PSR-3 (Logger Interface) 日志接口

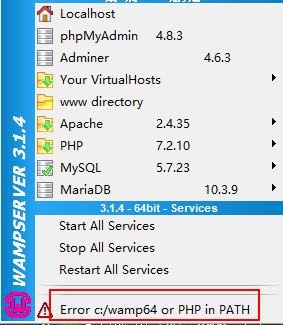
PSR-4 (Improved Autoloading) 自动加载优化标准

## 安装

Windows系统，下载运行 composer.exe，软件默认添加环境变量。

### 解决与wamp的PATH环境变量冲突

当使用wamp3.0版本，其集成多个php版本，composer选择其中一个php版本的路径添加的环境变量，导致开启wamp时，提示如下错误，影响wamp上的php版本切换：

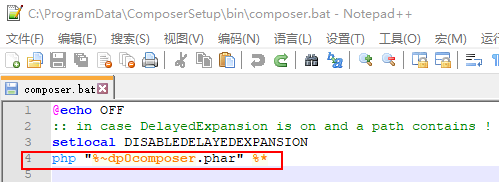


规避方法：

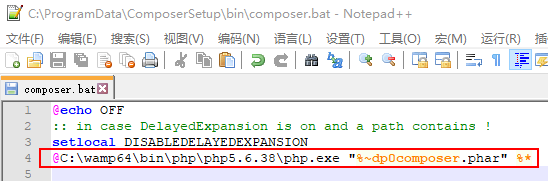
修改composer.bat来解决，直接修改C:\ProgramData\ComposerSetup\bin\composer.bat文件。

@C:\wamp64\bin\php\php5.6.38\php.exe "%~dp0composer.phar" %\*

修改前：



修改后：



验证：



## CN镜像

### 全局更换镜像

修改composer全局配置文件，cmd窗口执行：

composer config -g repo.packagist composer <https://packagist.phpcomposer.com>

### 当前项目更换

1. 修改当前项目的composer.json配置文件，在项目目录打开cmd窗口，执行

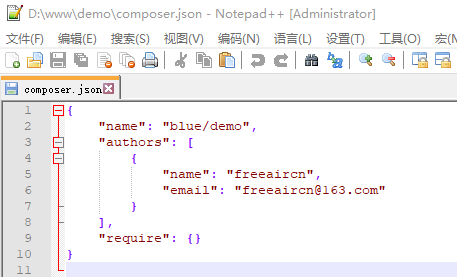
composer config repo.packagist composer <https://packagist.phpcomposer.com>

## 使用

### 初始化项目

1. 初始化工作目录，在项目的根目录命令行输入，composer init。

一些输入项可以先跳过，完成初始化工作后将在项目根目录生成composer.json文件



### 添加依赖包

1. 编辑composer.json文件，在require中写入依赖的包：



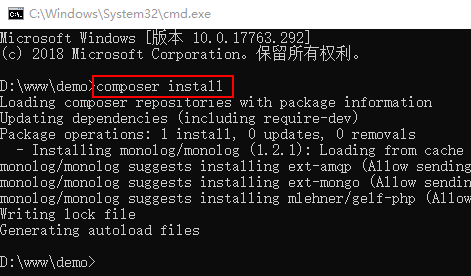
或者使用命令行，单独执行require和update

composer require monolog/monolog: 1.0.0

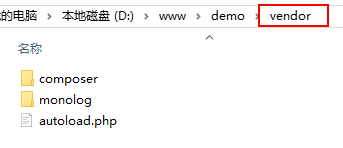
composer update monolog/monolog: 1.0.2

### 安装依赖包

1. 在composer.json同级目录下，执行composer install

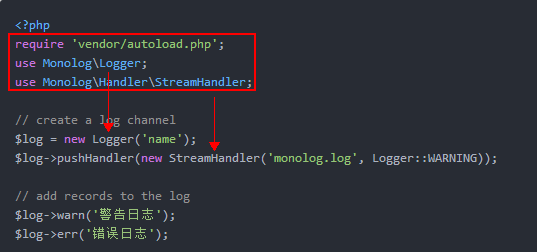


默认，在项目根目录下创建“vendor”文件夹，并将依赖包下载到vendor内，生成了autoload.php自动加载文件。



### 代码引用依赖包

1. 在vendor同级目录下创建代码文件，代码中引入autoload.php，即可使用第三方包。



### 更新项目依赖

1. 当项目根目录下已经生成composer.lock文件时，当执行命令composer intsall时composer只会下载composer.lock文件指定的版本，忽略 composer.json 文件中的定义，从而确保团队成员都使用相同的依赖包。

若package.json进行了修改需要更新依赖库和更新锁文件，可以执行以下命令：

composer update

## 自动加载/autoload

对于库的自动加载信息，Composer 生成了一个 vendor/autoload.php 文件。你可以简单的引入这个文件：

require 'vendor/autoload.php';

### PSR-4 autoloading

你可以在 composer.json 的 autoload 字段中增加自己的 autoloader。

{

"autoload": {

"psr-4": {"Acme\\": "src/"}

}

}

Composer 将注册一个 PSR-4 autoloader 到 Acme 命名空间。

你可以定义一个从命名空间到目录的映射。相对于项目的根目录，src与 vendor 文件夹同级。例如 src/Foo.php 文件应该包含 Acme\Foo 类。

When autoloading a class like Foo\\Bar\\Baz a namespace prefix Foo\\ pointing to a directory src/ means that the autoloader will look for a file named src/Bar/Baz.php and include it if present.

添加 autoload 字段后，你应该再次运行 install 命令来生成 vendor/autoload.php 文件。

The PSR-4 references are all combined, during install/update, into a single key => value array which may be found in the generated file vendor/composer/autoload\_psr4.php.

引用这个文件也将返回 autoloader 的实例，你可以将包含调用的返回值存储在变量中，并添加更多的命名空间。这对于在一个测试套件中自动加载类文件是非常有用的，例如。

$loader=require'vendor/autoload.php';

$loader->add('Acme\\Test\\',\_\_DIR\_\_);

### PSR-0

在 psr-0 key 下你定义了一个命名空间到实际路径的映射（相对于包的根目录）。注意，这里同样支持 PEAR-style 方式的约定（与命名空间不同，PEAR 类库在类名上采用了下划线分隔）。

请注意，命名空间的申明应该以 \\ 结束，以确保 autoloader 能够准确响应。例： Foo 将会与 FooBar 匹配，然而以反斜杠结束就可以解决这样的问题， Foo\\ 和 FooBar\\ 将会被区分开来。

在 install/update 过程中，PSR-0 引用都将被结合为一个单一的键值对数组，存储至 vendor/composer/autoload\_namespaces.php 文件中。

{

"autoload": {

"psr-0": {

"Monolog\\": "src/",

"Vendor\\Namespace\\": "src/",

"Vendor\_Namespace\_": "src/"

}

}

}

### Classmap

classmap 引用的所有组合，都会在 install/update 过程中生成，并存储到 vendor/composer/autoload\_classmap.php 文件中。这个 map 是经过扫描指定目录（同样支持直接精确到文件）中所有的 .php 和 .inc 文件里内置的类而得到的。

你可以用 classmap 生成支持支持自定义加载的不遵循 PSR-0/4 规范的类库。要配置它指向需要的目录，以便能够准确搜索到类文件。

实例：

{

"autoload": {

"classmap": ["src/", "lib/", "Something.php"]

}

}

### Files

如果你想要明确的指定，在每次请求时都要载入某些文件，那么你可以使用 'files' autoloading。通常作为函数库的载入方式（而非类库）。

实例：

{

"autoload": {

"files": ["src/MyLibrary/functions.php"]

}

}

## PHP命名空间

### 概述

在PHP中，命名空间用来解决在编写类库或应用程序时创建可重用的代码如类或函数时碰到的两类问题：

用户编写的代码与PHP内部的类/函数/常量或第三方类/函数/常量之间的名字冲突。

为很长的标识符名称(通常是为了缓解第一类问题而定义的)创建一个别名（或简短）的名称，提高源代码的可读性。

PHP 命名空间提供了一种将相关的类、函数和常量组合到一起的途径。

PHP5.3开始const关键字可以用在类的外部。const和define都是用来声明常量的（它们的区别不详述），但是在命名空间里，define的作用是全局的，而const则作用于当前空间。我在文中提到的常量是指使用const声明的常量。

### 基础

1. 创建一个命名空间需要使用namespace关键字，要注意的是，当前脚本文件的第一个命名空间前面不能有任何代码。

<?php

//创建一个名为'Article'的命名空间

namespace Article;

为什么要说第一个命名空间呢？因为同一脚本文件中可以创建多个命名空间。

<?php

//创建一个名为'Article'的命名空间

namespace Article;

//此Comment属于Article空间的元素

class Comment { }

//创建一个名为'MessageBoard'的命名空间

namespace MessageBoard;

//此Comment属于MessageBoard空间的元素

class Comment { }

1. 在不同空间之间不可以直接调用其它元素，需要使用命名空间的语法

<?php

namespace Article;

class Comment { }

namespace MessageBoard;

class Comment { }

//调用当前空间（MessageBoard）的Comment类

$comment = new Comment();

//调用Article空间的Comment类

$article\_comment = new \Article\Comment();

可以看到，在MessageBoard空间中调用article空间里的Comment类时，使用了一种像文件路径的语法： \空间名\元素名。

1. 除了类之外，对函数和常量的用法是一样的，下面我为两个空间创建了新的元素

<?php

namespace Article;

const PATH = '/article';

function getCommentTotal() {

return 100;

}

class Comment { }

namespace MessageBoard;

const PATH = '/message\_board';

function getCommentTotal() {

return 300;

}

class Comment { }

//调用当前空间的常量、函数和类

echo PATH; ///message\_board

echo getCommentTotal(); //300

$comment = new Comment();

//调用Article空间的常量、函数和类

echo \Article\PATH; ///article

echo \Article\getCommentTotal(); //100

$article\_comment = new \Article\Comment();

### 子空间

命名空间的调用语法像文件路径一样是有道理的，它允许我们自定义子空间来描述各个空间之间的关系。

<?php

//我用这样的命名空间表示处于blog下的article模块

namespace Blog\Article;

class Comment { }

//我用这样的命名空间表示处于blog下的message board模块

namespace Blog\MessageBoard;

class Comment { }

//调用当前空间的类

$comment = new Comment();

//调用Blog\Article空间的类

$article\_comment = new \Blog\Article\Comment();

而且，子空间还可以定义很多层次，比如说 Blog\Article\Archives\Date

### 公共空间

在一个命名空间里引入这个脚本，脚本里的元素不会归属到这个命名空间。如果这个脚本里没有定义其它命名空间，它的元素就始终处于公共空间中

<?php

namespace Blog\Article;

//引入脚本文件

include './common\_inc.php';

$filter\_XSS = new FilterXSS(); //出现致命错误：找不到Blog\Article\FilterXSS类

$filter\_XSS = new \FilterXSS(); //正确

调用公共空间的方式是直接在元素名称前加 \ 就可以了，否则PHP解析器会认为我想调用当前空间下的元素。除了自定义的元素，还包括PHP自带的元素，都属于公共空间。

要提一下，其实公共空间的函数和常量不用加 \ 也可以正常调用（不明白PHP为什么要这样做），但是为了正确区分元素，还是建议调用函数的时候加上 \

### 名称术语

在说别名和导入之前，需要知道关于空间三种名称的术语，以及PHP是怎样解析它们的。官方文档说得非常好，我就直接拿来套了。

1.非限定名称，或不包含前缀的类名称，例如 $comment = new Comment();。如果当前命名空间是Blog\Article，Comment将被解析为Blog\Article\Comment。如果使用Comment的代码不包含在任何命名空间中的代码（全局空间中），则Comment会被解析为Comment。

2.限定名称，或包含前缀的名称，例如 $comment = new Article\Comment();。如果当前的命名空间是Blog，则Comment会被解析为Blog\Article\Comment。如果使用Comment的代码不包含在任何命名空间中的代码（全局空间中），则Comment会被解析为Comment。

3.完全限定名称，或包含了全局前缀操作符的名称，例如 $comment = new \Article\Comment();。在这种情况下，Comment总是被解析为代码中的文字名(literal name)Article\Comment。

其实可以把这三种名称类比为文件名（例如 comment.php）、相对路径名（例如 ./article/comment.php）、绝对路径名（例如 /blog/article/comment.php），这样可能会更容易理解。

### 别名和导入

别名和导入可以看作是调用命名空间元素的一种快捷方式。PHP并不支持导入函数或常量。

<?php

namespace Blog\Article;

class Comment { }

//创建一个BBS空间（我有打算开个论坛）

namespace BBS;

//导入一个命名空间

use Blog\Article;

//导入命名空间后可使用限定名称调用元素

$article\_comment = new Article\Comment();

//为命名空间使用别名

use Blog\Article as Arte;

//使用别名代替空间名

$article\_comment = new Arte\Comment();

//导入一个类

use Blog\Article\Comment;

//导入类后可使用非限定名称调用元素

$article\_comment = new Comment();

//为类使用别名

use Blog\Article\Comment as Comt;

//使用别名代替空间名

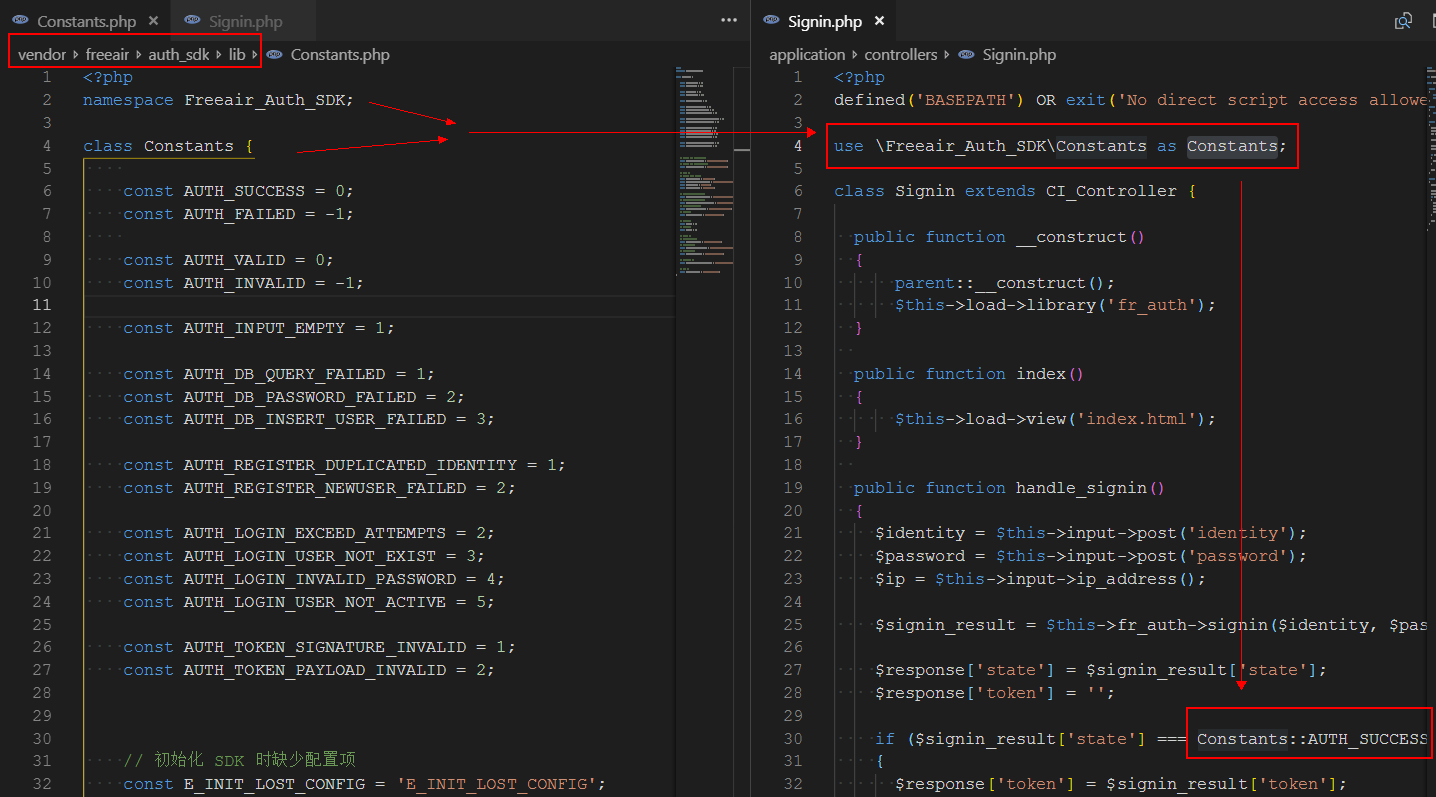
$article\_comment = new Comt();

## 使用举例

Load定义：



使用：



如果需要在建子目录，在子目录里建类，命名空间的规则应该是顶级命名空间\子目录\类名

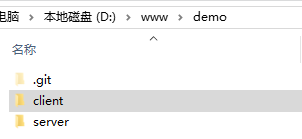


# 

# CodeIgniter和Vue框架

## 目录结构

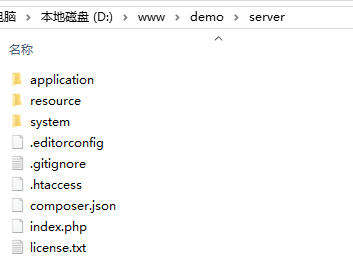
项目根目录demo，前端项目目录client，后端项目目录server。



## CodeIgniter框架

### 源文件路径

下载CI源文件后，存放于后端项目目录server下。



修改composer.json，去掉require-dev依赖包。



Composerinstall安装依赖包。

### 移除url中的index.php

借助apache的Rewrite模块功能，移除url中的index.php

1. 开启Apache的Rewrite模块

打开apache的配置文件，conf/httpd.conf :

LoadModule rewrite\_module modules/mod\_rewrite.so把该行前的#去掉。

虚拟主机路径，修改：AllowOverride All

1. 在CI的根目录下，即在index.php，system的同级目录下，创建文件.htaccess，复制如下内容：

RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

RewriteRule ^(.\*)$ index.php/$1 [L]

另外，index.php的同级目录下还有assets文件夹，这些需要过滤除去：

RewriteCond $1 !^(index\.php|images|assets|robots\.txt

1. 将CI中配置文件（application/config/config.php）中

$config['index\_page'] = "index.php";

改成

$config['index\_page'] = "";

重启apache，完成。

### 使用php命名空间

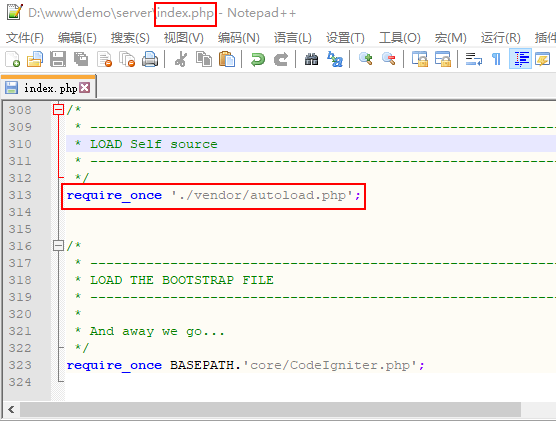
use ：只是进行声明就好像说明这里使用了某个命名空间下面的类，并不能实际载入该类。

require ：所以需要在 use之前，进行require

通过composer生成autoloader文件，引入该文件就能完成自动导入需要的类。

require 'vendor/autoload.php';

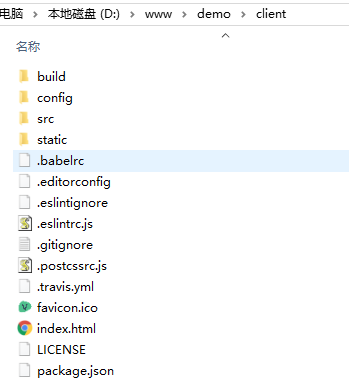
CI框架下，index.php是其入口文件，在index.php末尾引入'vendor/autoload.php'



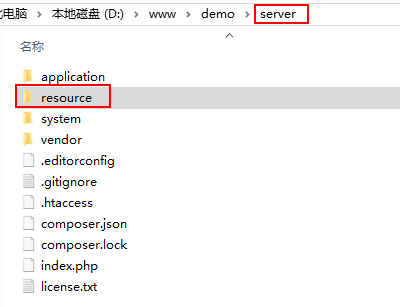
## Vue框架Element admin

### 源文件路径

1. 下载源文件后，存放于前端项目目录client下。



1. CI的目录下新建resource文件夹，存放前端打包输出的资源文件。



1. 下载element admin后，npm install安装项目依赖包。



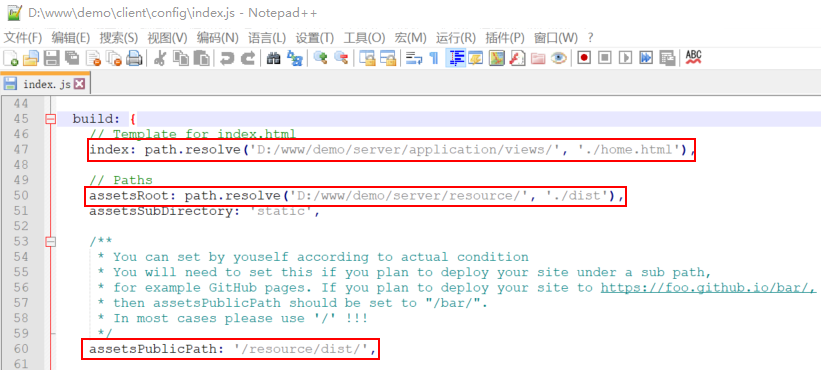
### Element admin build设置

1. 修改element admin的config文件：

首页视图文件（home.html）输出到CI的views目录下；取名home.html，因views目录下存在缺省index.html。

资源文件（js,css,ico…）输出到CI的resource目录下，全部存放于dist子文件夹；

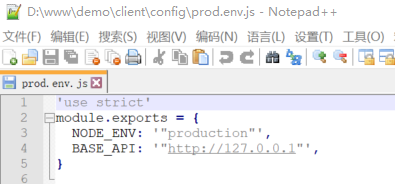
资源路径：'/resource/dist/'



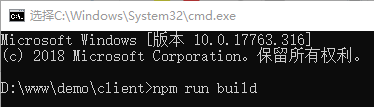
1. 修改pro的config文件：



部署服务器，使用服务器IP地址替换BASE\_API

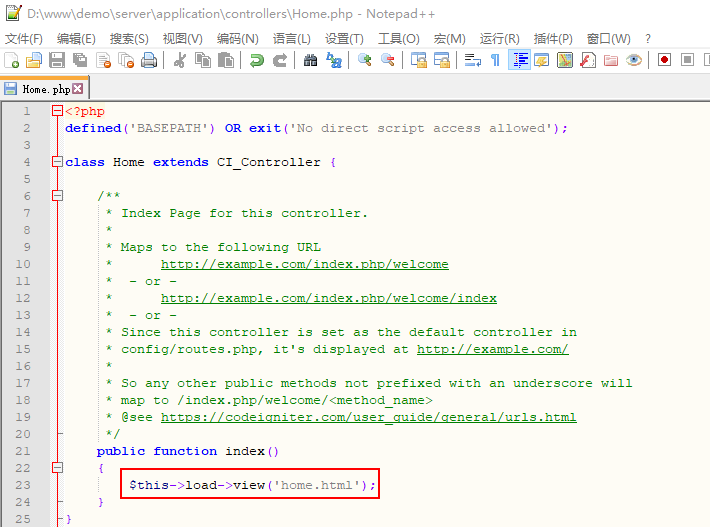


1. 生产element admin

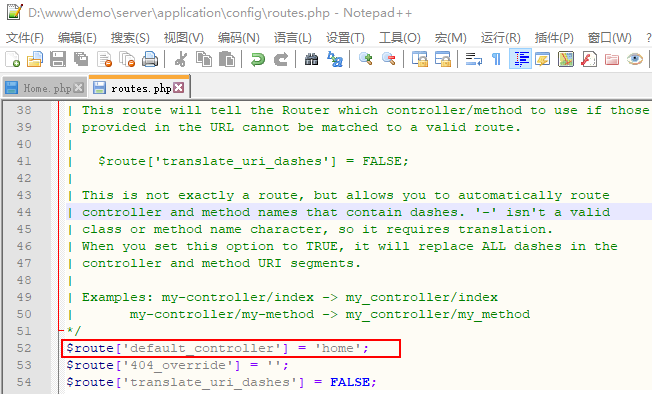


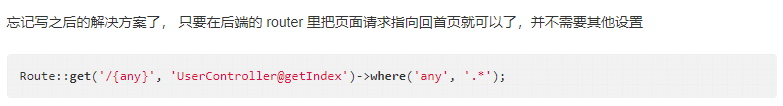
### 单页面HTML5 History的后端设置

1. 新建CI 控制器Home.php，视图加载element admin输出的home.html。



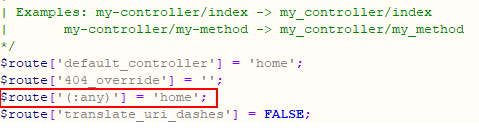
1. 调整CI的routers配置，default controller为Home.php





1. 在服务端增加一个覆盖所有情况的候选资源：如果 URL 匹配不到任何静态资源，则应该返回同一个html 页面，这个页面就是你 app 依赖的页面。

因为，vue打包输出页面文件为home.html，而CI的controller Home加载home.html视图，添加route：



1. 验证，打开默认主页

## CI项目引入iview admin

### CI目录添加iview-admin

1. CI根目录下新建resource文件夹，存放资源文件。



1. 将GitHub下载的iview-admin解压后的文件拷贝到resource文件夹。首次拷贝的iview，需要借助nmp install安装项目依赖包。



1. 验证iview admin：





### CI引入iviewadmin

1. 修改iview admin的vue.config.js文件：



1. 生产iview admin



1. CI index.php文件，view folder填写iview admin的dist绝对路径。



1. 修改CI 控制器welcome.php，视图加载iview admin的dist中的index.html。



1. 验证，通过CI打开iview admin默认主页



# SCSS

## 概述

1. sass是由buby语言编写的一款css预处理语言，和html一样有严格的缩进风格，和css编写规范有着很大的出入，是不使用花括号和分号的，所以不被广为接受。
2. sass和scss其实是一样的css预处理语言，其后缀名是分别为 .sass和.scss两种。

SASS版本3.0之前的后缀名为.sass，而版本3.0之后的后缀名.scss。

两者是有不同的，继sass之后scss的编写规范基本和css一致，sass时代是有严格的缩进规范并且没有‘{}’和‘；’。而scss则和css的规范是一致的。

1. Sass是成熟、稳定、强大的CSS预处理器，而SCSS是Sass3版本当中引入的新语法特性，完全兼容CSS3的同时继承了Sass强大的动态功能。

通过SASS预处理器进行CSS的开发，SASS提供的变量、嵌套、混合、继承等特性，让CSS的书写更加有趣与程式化。

## 变量

1. 变量用来存储需要在CSS中复用的信息，例如颜色和字体。SASS通过$符号去声明一个变量。

$font-stack: Helvetica, sans-serif;

$primary-color: #333;

body {

font: 100% $font-stack;

color: $primary-color;

}

## 嵌套

1. 以嵌套的方式使用CSS，但是过度的使用嵌套会让产生的CSS难以维护，因此是一种不好的实践，下面的例子表达了一个典型的网站导航样式。

nav {

ul {

margin: 0;

padding: 0;

list-style: none;

}

li { display: inline-block; }

}

编译后产生的CSS代码如下

navul {

margin: 0;

padding: 0;

list-style: none; }

navli {

display: inline-block; }

## 引入/@import

1. SASS能够将代码分割为多个片段，并以underscore风格的下划线作为其命名前缀（\_partial.scss），SASS会通过这些下划线来辨别哪些文件是SASS片段，并且不让片段内容直接生成为CSS文件，从而只是在使用@import指令的位置被导入。CSS原生的@import会通过额外的HTTP请求获取引入的样式片段，而SASS的@import则会直接将这些引入的片段合并至当前CSS文件，并且不会产生新的HTTP请求。下面例子中的代码，将会在base.scss文件当中引入\_reset.scss片断。

scss引用的外部文件命名必须以\_开头, 如下例所示:其中\_test1.scss、\_test2.scss、\_test3.scss文件分别设置的h1 h2 h3。文件名如果以下划线\_开头的话，Sass会认为该文件是一个引用文件，不会将其编译为css文件

// \_reset.scss

html, body, ul, ol {

margin: 0;

padding: 0;

}

// base.scss

@import 'reset';

body {

font: 100% Helvetica, sans-serif;

background-color: #efefef;

}

SASS中引入片断时，可以缺省使用文件扩展名，因此上面代码中直接通过@import 'reset'引入，编译后生成的代码如下

html, body, ul, ol {

margin: 0;

padding: 0; }

body {

font: 100% Helvetica, sans-serif;

background-color: #efefef; }

## 混合/@mixin

1. 混合（Mixin）用来分组那些需要在页面中复用的CSS声明，开发人员可以通过向Mixin传递变量参数来让代码更加灵活，该特性在添加浏览器兼容性前缀的时候非常有用，SASS目前使用@mixin name指令来进行混合操作。

@mixin border-radius($radius) {

border-radius: $radius;

-ms-border-radius: $radius;

-moz-border-radius: $radius;

-webkit-border-radius: $radius;

}

.box {

@include border-radius(10px);

}

上面的代码建立了一个名为border-radius的Mixin，并传递了一个变量$radius作为参数，然后在后续代码中通过@include border-radius(10px)使用该Mixin。

## 继承/@extend

1. 继承是SASS中非常重要的一个特性，可以通过@extend指令在选择器之间复用CSS属性，并且不会产生冗余的代码，下面例子将会通过SASS提供的继承机制建立一系列样式。

// 这段代码不会被输出到最终生成的CSS文件，因为它没有被任何代码所继承。

%other-styles {

display: flex;

flex-wrap: wrap;

}

// 下面代码会正常输出到生成的CSS文件，因为它被其接下来的代码所继承。

%message-common {

border: 1px solid #ccc;

padding: 10px;

color: #333;

}

.message {

@extend %message-common;

}

.success {

@extend %message-common;

border-color: green;

}

## 操作符

1. SASS提供了标准的算术运算符，例如+、-、\*、/、%。

## CSS扩展

### 引用符级选择器

1. Scss使用"&"关键字在CSS规则中引用父级选择器，例如在嵌套使用伪类选择器的场景下

/\*===== SCSS =====\*/

a {

font-weight: bold;

text-decoration: none;

&:hover { text-decoration: underline; }

body.firefox & { font-weight: normal; }

}

/\*===== CSS =====\*/

a {

font-weight: bold;

text-decoration: none; }

a:hover {

text-decoration: underline; }

body.firefox a {

font-weight: normal; }

1. 无论CSS规则嵌套的深度怎样，关键字"&"都会使用父级选择器级联替换全部其出现的位置：

/\*===== SCSS =====\*/

#main {

color: black;

a {

font-weight: bold;

&:hover { color: red; }

}

}

/\*===== CSS =====\*/

#main {

color: black; }

#main a {

font-weight: bold; }

#main a:hover {

color: red; }

1. "&"必须出现在复合选择器开头的位置，后面再连接自定义的后缀，例如：

/\*===== SCSS =====\*/

#main {

color: black;

&-sidebar { border: 1px solid; }

}

/\*===== CSS =====\*/

#main {

color: black; }

#main-sidebar {

border: 1px solid; }

### 嵌套属性

1. CSS许多属性都位于相同的命名空间（例如font-family、font-size、font-weight都位于font命名空间下），Scss当中只需要编写命名空间一次，后续嵌套的子属性都将会位于该命名空间之下，请看下面的代码。命令空间后带有冒号:

/\*===== SCSS =====\*/

.demo {

// 命令空间后带有冒号:

font: {

family: fantasy;

size: 30em;

weight: bold;

}

}

/\*===== CSS =====\*/

.demo {

font-family: fantasy;

font-size: 30em;

font-weight: bold; }

1. 命令空间上可以直接书写CSS简写属性。

.demo {

font: 20px/24px fantasy {

weight: bold;

}

}

.demo {

font: 20px/24px fantasy;

font-weight: bold;

}

# 

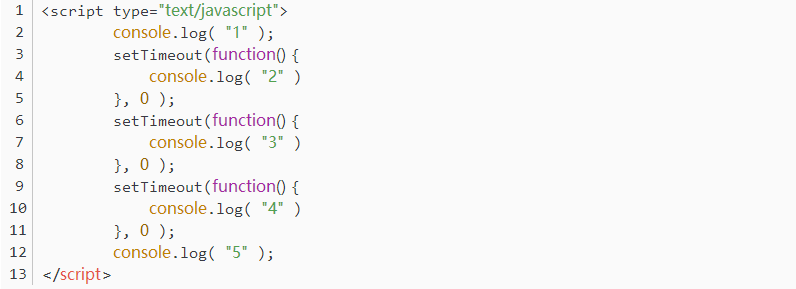
# JS异步

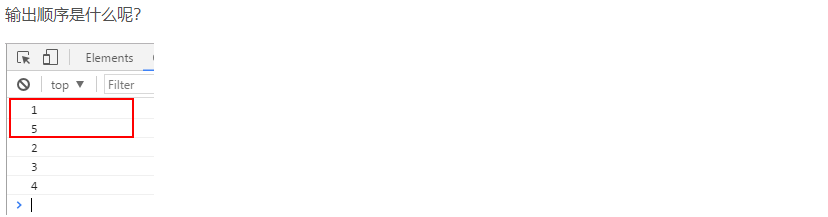
## JS的同步和异步

1. A

javascript语言是一门“单线程”的语言，不像java语言，类继承Thread再来个thread.start就可以开辟一个线程，所以，javascript就像一条流水线，仅仅是一条流水线而已，要么加工，要么包装，不能同时进行多个任务和流程。

最基础的异步是setTimeout和setInterval函数，很常见，但是很少人有人知道其实这就是异步，因为它们可以控制js的执行顺序。我们也可以简单地理解为：可以改变程序正常执行顺序的操作就可以看成是异步操作。





火狐浏览器的api文档有这样一句话：Because even though setTimeout was called with a delay of zero, it's placed on a queue and scheduled to run at the next opportunity, not immediately. Currently executing code must complete before functions on the queue are executed, the resulting execution order may not be as expected

javascript是单线程。单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等着。于是就有一个概念——任务队列。如果排队是因为计算量大，CPU忙不过来，倒也算了，但是很多时候CPU是闲着的，因为IO设备（输入输出设备）很慢（比如Ajax操作从网络读取数据），不得不等着结果出来，再往下执行。于是JavaScript语言的设计者意识到，这时主线程完全可以不管IO设备，挂起处于等待中的任务，先运行排在后面的任务。等到IO设备返回了结果，再回过头，把挂起的任务继续执行下去。

于是，所有任务可以分成两种，一种是同步任务（synchronous），另一种是异步任务（asynchronous）。同步任务指的是，在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务；异步任务指的是，不进入主线程、而进入"任务队列"（task queue）的任务，只有等主线程任务执行完毕，"任务队列"开始通知主线程，请求执行任务，该任务才会进入主线程执行。

具体来说，异步运行机制如下：

（1）所有同步任务都在主线程上执行，形成一个执行栈（execution context stack）。

（2）主线程之外，还存在一个"任务队列"（task queue）。只要异步任务有了运行结果，就在"任务队列"之中放置一个事件。

（3）一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。

（4）主线程不断重复上面的第三步。

只要主线程空了，就会去读取"任务队列"。

"任务队列"是一个事件的队列（也可以理解成消息的队列），IO设备完成一项任务，就在"任务队列"中添加一个事件，表示相关的异步任务可以进入"执行栈"了。主线程读取"任务队列"，就是读取里面有哪些事件。

"任务队列"中的事件，除了IO设备的事件以外，还包括一些用户产生的事件（比如鼠标点击、页面滚动等等），比如$(selectot).click(function)，这些都是相对耗时的操作。只要指定过这些事件的回调函数，这些事件发生时就会进入"任务队列"，等待主线程读取。

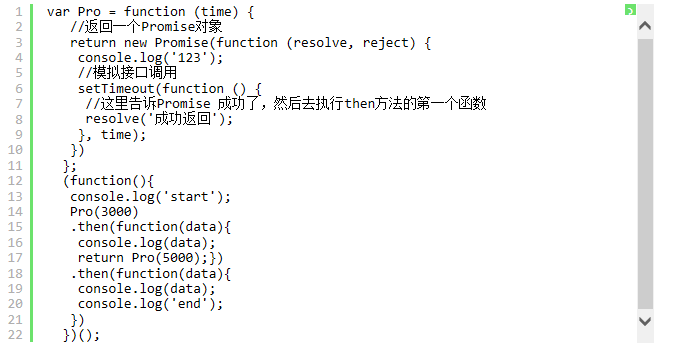
所谓"回调函数"（callback），就是那些会被主线程挂起来的代码，前面说的点击事件$(selectot).click(function)中的function就是一个回调函数。异步任务必须指定回调函数，当主线程开始执行异步任务，就是执行对应的回调函数。例如ajax的success，complete，error也都指定了各自的回调函数，这些函数就会加入“任务队列”中，等待执行。

## Promise

1. A

Promise（ES6），是一个对象，是用来处理异步操作的，可以让我们写异步调用的时候写起来更加优雅，更加美观便于阅读。顾名思义为承诺、许诺的意思，意思是使用了Promise之后他肯定会给我们答复，无论成功或者失败都会给我们一个答复。所以，Promise有三种状态：pending（进行中），resolved（完成），rejected（失败）。只有异步返回的结果可以改变其状态。所以，promise的过程一般只有两种：pending->resolved或者pending->rejected。

promise对象还有一个比较常用的then方法，用来执行回调函数，then方法接受两个参数，第一个是成功的resolved的回调，另一个是失败rejected的回调，第二个失败的回调参数可选。并且then方法里也可以返回promise对象，这样就可以链式调用了。

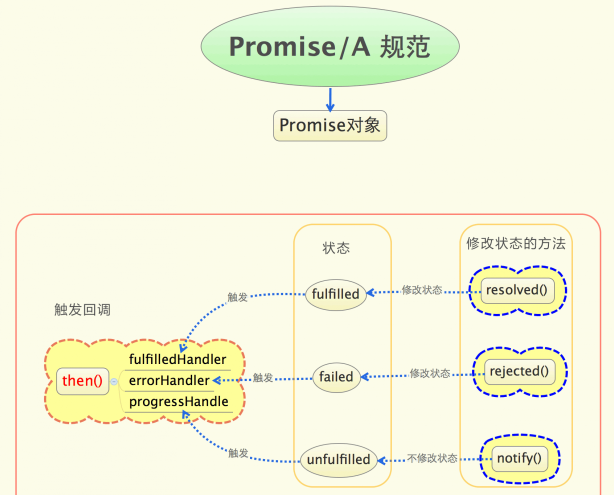


上边代码中，定义了一个Pro变量，然后把一个匿名函数赋给他，函数返回一个Promise对象，然后对象里边接收一个函数，分别把resolve跟reject方法当参数传进去，用setTimeOut来模拟异步请求，当执行resolve方法后就会调用then方法的一个函数。结果如下



promise必须实现then方法（可以说，then就是promise的核心），而且then必须返回一个promise，同一个promise的then可以调用多次，并且回调的执行顺序跟它们被定义时的顺序一致

then方法接受两个参数，第一个参数是成功时的回调，在promise由“等待”态转换到“完成”态时调用，另一个是失败时的回调，在promise由“等待”态转换到“拒绝”态时调用。同时，then可以接受另一个promise传入，也接受一个“类then”的对象或方法，即thenable对象。



//



## Axios

1. A

axios 是一个基于Promise 用于浏览器和 nodejs 的 HTTP 客户端，它本身具有以下特征：

从浏览器中创建 XMLHttpRequest

从 node.js 发出 http 请求

支持 Promise API

拦截请求和响应

转换请求和响应数据

取消请求

自动转换JSON数据

客户端支持防止 CSRF/XSRF

### 请求配置选项

1. A

创建请求时可以用的配置选项。只有 url 是必需的。如果没有指定 method，请求将默认使用 get 方法。

{

// `url` 是用于请求的服务器 URL

url: '/user',

// `method` 是创建请求时使用的方法

method: 'get', // 默认是 get

// `baseURL` 将自动加在 `url` 前面，除非 `url` 是一个绝对 URL。

// 它可以通过设置一个 `baseURL` 便于为 axios 实例的方法传递相对 URL

baseURL: 'https://some-domain.com/api/',

// `transformRequest` 允许在向服务器发送前，修改请求数据

// 只能用在 'PUT', 'POST' 和 'PATCH' 这几个请求方法

// 后面数组中的函数必须返回一个字符串，或 ArrayBuffer，或 Stream

transformRequest: [function (data) {

// 对 data 进行任意转换处理

return data;

}],

// `transformResponse` 在传递给 then/catch 前，允许修改响应数据

transformResponse: [function (data) {

// 对 data 进行任意转换处理

return data;

}],

// `headers` 是即将被发送的自定义请求头

headers: {'X-Requested-With': 'XMLHttpRequest'},

// `params` 是即将与请求一起发送的 URL 参数

// 必须是一个无格式对象(plain object)或 URLSearchParams 对象

params: {

ID: 12345

},

// `paramsSerializer` 是一个负责 `params` 序列化的函数

// (e.g. https://www.npmjs.com/package/qs, http://api.jquery.com/jquery.param/)

paramsSerializer: function(params) {

return Qs.stringify(params, {arrayFormat: 'brackets'})

},

// `data` 是作为请求主体被发送的数据

// 只适用于这些请求方法 'PUT', 'POST', 和 'PATCH'

// 在没有设置 `transformRequest` 时，必须是以下类型之一：

// - string, plain object, ArrayBuffer, ArrayBufferView, URLSearchParams

// - 浏览器专属：FormData, File, Blob

// - Node 专属： Stream

data: {

firstName: 'Fred'

},

// `timeout` 指定请求超时的毫秒数(0 表示无超时时间)

// 如果请求话费了超过 `timeout` 的时间，请求将被中断

timeout: 1000,

// `withCredentials` 表示跨域请求时是否需要使用凭证

withCredentials: false, // 默认的

// `adapter` 允许自定义处理请求，以使测试更轻松

// 返回一个 promise 并应用一个有效的响应 (查阅 [response docs](#response-api)).

adapter: function (config) {

/\* ... \*/

},

// `auth` 表示应该使用 HTTP 基础验证，并提供凭据

// 这将设置一个 `Authorization` 头，覆写掉现有的任意使用 `headers` 设置的自定义 `Authorization`头

auth: {

username: 'janedoe',

password: 's00pers3cret'

},

// `responseType` 表示服务器响应的数据类型，可以是 'arraybuffer', 'blob', 'document', 'json', 'text', 'stream'

responseType: 'json', // 默认的

// `xsrfCookieName` 是用作 xsrf token 的值的cookie的名称

xsrfCookieName: 'XSRF-TOKEN', // default

// `xsrfHeaderName` 是承载 xsrf token 的值的 HTTP 头的名称

xsrfHeaderName: 'X-XSRF-TOKEN', // 默认的

// `onUploadProgress` 允许为上传处理进度事件

onUploadProgress: function (progressEvent) {

// 对原生进度事件的处理

},

// `onDownloadProgress` 允许为下载处理进度事件

onDownloadProgress: function (progressEvent) {

// 对原生进度事件的处理

},

// `maxContentLength` 定义允许的响应内容的最大尺寸

maxContentLength: 2000,

// `validateStatus` 定义对于给定的HTTP 响应状态码是 resolve 或 reject promise 。如果 `validateStatus` 返回 `true` (或者设置为 `null` 或 `undefined`)，promise 将被 resolve; 否则，promise 将被 rejecte

validateStatus: function (status) {

return status >= 200 && status < 300; // 默认的

},

// `maxRedirects` 定义在 node.js 中 follow 的最大重定向数目

// 如果设置为0，将不会 follow 任何重定向

maxRedirects: 5, // 默认的

// `httpAgent` 和 `httpsAgent` 分别在 node.js 中用于定义在执行 http 和 https 时使用的自定义代理。允许像这样配置选项：

// `keepAlive` 默认没有启用

httpAgent: new http.Agent({ keepAlive: true }),

httpsAgent: new https.Agent({ keepAlive: true }),

// 'proxy' 定义代理服务器的主机名称和端口

// `auth` 表示 HTTP 基础验证应当用于连接代理，并提供凭据

// 这将会设置一个 `Proxy-Authorization` 头，覆写掉已有的通过使用 `header` 设置的自定义 `Proxy-Authorization` 头。

proxy: {

host: '127.0.0.1',

port: 9000,

auth: : {

username: 'mikeymike',

password: 'rapunz3l'

}

},

// `cancelToken` 指定用于取消请求的 cancel token

// （查看后面的 Cancellation 这节了解更多）

cancelToken: new CancelToken(function (cancel) {

})

}

### 请求方法别名

1. A

axios.request(config)

axios.get(url[, config])

axios.delete(url[, config])

axios.head(url[, config])

axios.options(url[, config])

axios.post(url[, data[, config]])

axios.put(url[, data[, config]])

axios.patch(url[, data[, config]])

注释

当使用以上别名方法时，url，method和data等属性不用在config重复声明。

### 响应结构

1. A

某个请求的响应包含以下信息

{

// `data` 由服务器提供的响应

data: {},

// `status` 来自服务器响应的 HTTP 状态码

status: 200,

// `statusText` 来自服务器响应的 HTTP 状态信息

statusText: 'OK',

// `headers` 服务器响应的头

headers: {},

// `config` 是为请求提供的配置信息

config: {}

}

使用 then 时，你将接收下面这样的响应：

axios.get('/user/12345')

.then(function(response) {

console.log(response.data);

console.log(response.status);

console.log(response.statusText);

console.log(response.headers);

console.log(response.config);

});

### 配置

#### 更改全局默认配置

1. A

axios.defaults.baseURL = 'https://api.example.com';

axios.defaults.headers.common['Authorization'] = AUTH\_TOKEN;

axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';

#### 自定义实例配置

1. A

// 创建实例时设置配置的默认值

var instance = axios.create({

baseURL: 'https://api.example.com'

});

// 在实例已创建后修改默认值

instance.defaults.headers.common['Authorization'] = AUTH\_TOKEN;

#### 配置的优先顺序

1. A

配置会以一个优先顺序进行合并。这个顺序是：在 lib/defaults.js 找到的库的默认值，然后是实例的 defaults 属性，最后是请求的 config 参数。后者将优先于前者。

### 拦截器

1. A

在请求或响应被 then 或 catch 处理前拦截它们。

// 添加请求拦截器

axios.interceptors.request.use(function (config) {

// 在发送请求之前做些什么

return config;

}, function (error) {

// 对请求错误做些什么

return Promise.reject(error);

});

// 添加响应拦截器

axios.interceptors.response.use(function (response) {

// 对响应数据做点什么

return response;

}, function (error) {

// 对响应错误做点什么

return Promise.reject(error);

});

如果你想在稍后移除拦截器，可以这样：

var myInterceptor = axios.interceptors.request.use(function () {/\*...\*/});

axios.interceptors.request.eject(myInterceptor);

可以为自定义 axios 实例添加拦截器

var instance = axios.create();

instance.interceptors.request.use(function () {/\*...\*/});

### 请求数据序列化

1. A

这就是使用axios和jquery ajax最大的区别。

jquery提交数据的时候，默认是以FormData的形式提交的，也就是Content-Type:"application/x-www-form-urlencoded",

而默认axios是使用的是Payload形式提交数据，也就是Content-Type:"application/json"

Using application/x-www-form-urlencoded format

By default, axios serializes JavaScript objects to JSON. To send data in the application/x-www-form-urlencoded format instead, you can use one of the following options.

Alternatively, you can encode data using the qs library:

var qs = require('qs');

axios.post('/foo', qs.stringify({ 'bar': 123 }));

### 错误处理

1. A

axios.get('/user/12345')

.catch(function (error) {

if (error.response) {

// 请求已发出，但服务器响应的状态码不在 2xx 范围内

console.log(error.response.data);

console.log(error.response.status);

console.log(error.response.headers);

} else {

// Something happened in setting up the request that triggered an Error

console.log('Error', error.message);

}

console.log(error.config);

});

可以使用 validateStatus 配置选项定义一个自定义 HTTP 状态码的错误范围。

axios.get('/user/12345', {

validateStatus: function (status) {

return status < 500; // 状态码在大于或等于500时才会 reject

}

})

# 附录

## A

### 

CSS 命名采用 BEM 的风格