|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | | |  |
|  | | |
|  | | |
| 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](#_Toc535850923)

[2 Windows设置 1](#_Toc535850924)

[2.1 右键菜单添加打开cmd 1](#_Toc535850925)

[3 文本编辑器 2](#_Toc535850926)

[3.1 安装Notepad++ 2](#_Toc535850927)

[3.2 安装VS Code 2](#_Toc535850928)

[3.2.1 转换为中文 2](#_Toc535850929)

[3.2.2 插件 2](#_Toc535850930)

[3.3 安装Sublime Text（可选） 3](#_Toc535850931)

[3.3.1 授权 3](#_Toc535850932)

[3.3.2 修改Tab字符数 4](#_Toc535850933)

[3.3.3 解决Package Control下载失败问题 4](#_Toc535850934)

[3.3.4 解决Install Package下载失败问题 5](#_Toc535850935)

[3.3.5 插件 5](#_Toc535850936)

[4 NPM 7](#_Toc535850937)

[4.1 简介 7](#_Toc535850938)

[4.2 安装Node.js 7](#_Toc535850939)

[4.3 修改npm模块路径和cache路径 7](#_Toc535850940)

[4.4 安装cnpm 9](#_Toc535850941)

[4.5 删除模块 9](#_Toc535850942)

[5 Webpack 10](#_Toc535850943)

[5.1 简介 10](#_Toc535850944)

[5.2 安装webpack 10](#_Toc535850945)

[5.3 安装加载器/Loader 10](#_Toc535850946)

[5.4 安装插件/Plugin 11](#_Toc535850947)

[6 Git 12](#_Toc535850948)

[6.1 关联用户信息 12](#_Toc535850949)

[6.2 创建本地仓库 12](#_Toc535850950)

[6.3 本地仓库添加文件 12](#_Toc535850951)

[6.4 本地仓库提交文件 12](#_Toc535850952)

[6.5 分支 12](#_Toc535850953)

[7 Github 13](#_Toc535850954)

[7.1 添加hosts 13](#_Toc535850955)

[7.2 SSH配置 13](#_Toc535850956)

[7.3 gitignore忽略文件 13](#_Toc535850957)

[7.3.1 忽略优先级 13](#_Toc535850958)

[7.3.2 忽略规则 13](#_Toc535850959)

[7.3.3 注意 15](#_Toc535850960)

[7.4 新建远程仓库 15](#_Toc535850961)

[7.5 本地仓库与远程仓库关联 15](#_Toc535850962)

[7.6 本地仓库推送至远程仓库 15](#_Toc535850963)

[7.7 克隆远程仓库 16](#_Toc535850964)

[7.8 从远程仓库更新本地仓库 16](#_Toc535850965)

[8 WAMP 17](#_Toc535850966)

[8.1 下载地址 17](#_Toc535850967)

[8.2 修改mysql的root密码 17](#_Toc535850968)

[8.3 修改wampserver的www路径 18](#_Toc535850969)

[8.4 添加virtual host 18](#_Toc535850970)

[8.5 Apache允许其他主机访问项目目录 18](#_Toc535850971)

[8.6 修改phpadmin登录方式 19](#_Toc535850972)

[9 PHP Composer 20](#_Toc535850973)

[9.1 简介 20](#_Toc535850974)

[9.2 安装 20](#_Toc535850975)

[9.3 CN镜像 20](#_Toc535850976)

[9.4 composer.json 20](#_Toc535850977)

[9.5 安装依赖包 21](#_Toc535850978)

[9.6 自动加载 21](#_Toc535850979)

[10 CodeIgniter和admin框架适配 23](#_Toc535850980)

[10.1 CI初始配置 23](#_Toc535850981)

[10.2 CI去掉默认url中的index.php 24](#_Toc535850982)

[10.3 CI项目引入iview admin 26](#_Toc535850983)

[10.3.1 CI目录添加iview-admin 26](#_Toc535850984)

[10.3.2 CI引入iview admin 28](#_Toc535850985)

[10.4 CI项目引入element admin 30](#_Toc535850986)

[10.4.1 CI目录添加element admin 30](#_Toc535850987)

[10.4.2 CI引入element admin 32](#_Toc535850988)

[11 附录 36](#_Toc535850989)

[11.1 A 36](#_Toc535850990)

# 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

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

名字: icmd

默认值: 在此处打开命令行 (注: 默认值可以自行设置)

1. 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

1. Ctrl+shift+P,弹出命令行，选择"Configure Display Language"



1. 安装“Chinese”插件



1. 重启VS Code

### 插件

1. 名称：Eslint

eslint检查JavaScript 代码是否满足指定规则的静态代码检查工具。

统一代码风格规则，如：缩进用几个空格；是否用驼峰命名法来命名变量和函数名等。

减少错误， 如：相等比较必须用 ===，变量在使用前必须被声明，在条件语句中不能使用赋值语句等。

提高代码质量，如：函数最多有多少条件分支；最多有几个参数，代码块最多能嵌套多少层等。

其他。如： 禁用alert。这可以提高用户体验，因为 alert 框的外观不是那么好看，而且往往与网站的风格不搭，一般都会自定义 alert 框。

1. 名称：Auto Close Tag，自动闭合HTML标签
2. 名称：Auto Rename Tag，修改HTML标签时，自动修改匹配的标签
3. 名称：Beautify css/sass/scss/less css/sass/less格式化
4. 名称：Better Align 对齐赋值符号和注释

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+cmd+=", "command": "wwm.aligncode",

"when": "editorTextFocus && !editorReadonly" }

1. 名称：Better Comments，编写更加人性化的注释
2. 名称：Bookmarks，添加行书签
3. 名称：Bracket Pair Colorizer，用不同颜色高亮显示匹配的括号
4. 名称：gitignore，.gitignore文件语法
5. 名称：Prettify JSON，格式化JSON
6. 名称：String Manipulation，字符串转换处理（驼峰、大写开头、下划线等等）
7. 名称：TODO Parser，Todo管理
8. 名称：Vue TypeScript Snippets，Vue Typescript代码段
9. 名称：VueHelper，Vue2代码段（包括Vue2 api、vue-router2、vuex2）
10. 名称：Vetur
11. 名称：phpfmt，自动格式化php代码：phpfmt - PHP formatter
12. 名称：PHP DocBlocker，生成函数的注释（/\*\*）

## 安装Sublime Text（可选）

### 授权

1. hosts文件添加

127.0.0.1 license.sublimehq.com

127.0.0.1 45.55.255.55

127.0.0.1 45.55.41.223

1. Sublime中输入license

----- BEGIN LICENSE -----

sgbteam

Single User License

EA7E-1153259

8891CBB9 F1513E4F 1A3405C1 A865D53F

115F202E 7B91AB2D 0D2A40ED 352B269B

76E84F0B CD69BFC7 59F2DFEF E267328F

215652A3 E88F9D8F 4C38E3BA 5B2DAAE4

969624E7 DC9CD4D5 717FB40C 1B9738CF

20B3C4F1 E917B5B3 87C38D9C ACCE7DD8

5F7EF854 86B9743C FADC04AA FB0DA5C0

F913BE58 42FEA319 F954EFDD AE881E0B

------ END LICENSE ------

### 修改Tab字符数

"tab\_size": 2,

"translate\_tabs\_to\_spaces": true,



### 解决Package Control下载失败问题

sublime text 的package control 无法安装插件。



解决方法：

Package Control.sublime-settings]修改方法：

Preferences > Package Settings > Package Control > Settings - User添加

"channels":

[

"http://cst.stu.126.net/u/json/cms/channel\_v3.json",

//"https://packagecontrol.io/channel\_v3.json",

//"https://web.archive.org/web/20160103232808/https://packagecontrol.io/channel\_v3.json",

//"https://gist.githubusercontent.com/nick1m/660ed046a096dae0b0ab/raw/e6e9e23a0bb48b44537f61025fbc359f8d586eb4/channel\_v3.json"

],





### 解决Install Package下载失败问题



到github或第三方网站手动下载安装包，然后解压到安装目录下的/Packages目录下

### 插件

1. 名称：BracketHighlighter -

匹配括号，引号等符号内的范围

1. 名称：Alignment -

使用方法：选中要调整的行，然后按 Ctrl + alt + a

1. 名称：All Autocomplete

搜索所有打开的文件来寻找匹配的提示词

1. 名称：AutoFileName

自动补全文件路径和文件名

1. 名称：Emmet -

提高HTML/CSS代码编写的速度。

1. 名称：DocBlockr -

自动补全注释插件: 1）输入/\*，回车或Tab；2）在函数的上面输入/\*\*，回车或Tab。

支持JavaScript (including ES6), PHP, ActionScript, Haxe, CoffeeScript, TypeScript, Java, Apex, Groovy, Objective C, C, C++ and Rust.等众多语言

1. 名称：JsFormat -

使用方法：在JS文件中通过鼠标右键->JsFormat或快捷键 ctrl + alt + f 对JS进行格式化

1. 名称：HTML-CSS-JS Prettify -

使用方法：在文件Ctrl+Shift+h。（缩进对齐）

1. 名称：SideBarEnhancements -

增强文件夹栏右键菜单

1. 名称：SublimeTmpl -

新建 html、css、javascript、php、python、ruby 六种类型的文件模板

1. 名称：Vue syntax highlight -

Vue语法高亮显示

1. 名称：Pretty Json -
2. 使用方法：ctrl + shift + j

# NPM

## 简介

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

1. 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.



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

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



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



1. 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；

# Webpack

## 简介

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！这就是它不可不说的优点，包括你的JavaScript代码，也包括CSS和fonts以及图片等等等，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，插件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

cnpm install webpack -g

cnpm install webpack-cli -g

## 安装加载器/Loader

1. CSS 样式Loader

cnpm install css-loader -g

cnpm install style-loader -g

1. 图片和字体等Loader

cnpm install url-loader -g

cnpm install file-loader -g

## 安装插件/Plugin

1. extract-text-webpack-plugin

cnpm install extract-text-webpack-plugin -g

1. 打包依赖插件

cnpm install webpack-merge -g

cnpm install html-webpack-plugin -g

# Git

## 关联用户信息

1. 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

1. 切换到保存仓库目录
2. 命令：git init

## 本地仓库添加文件

1. 添加指定文件：git add filename
2. 添加当前路径下所有文件：git add .

## 本地仓库提交文件

1. 命令：git commit -m "备注"

## 分支

1. 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

1. 合并分支

git merge分支名

1. 删除分支

git branch -d 分支名

1. 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。

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

id\_rsa是私钥，id\_rsa.pub是公钥

1. 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）：

* 1. 从命令行中读取可用的忽略规则
  2. 当前目录定义的规则
  3. 父级目录定义的规则，依次递推
  4. $GIT\_DIR/info/exclude 文件中定义的规则
  5. core.excludesfile中定义的全局规则配置运维总结

### 忽略规则

1. 空格不匹配任意文件，可作为分隔符，可用反斜杠转义
2. 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。
3. 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。
4. 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。

1. 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。
2. 以问号"?"通配单个字符，即匹配一个任意字符；
3. 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）。
4. 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，否则一单push，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

1. 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

1. 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

1. 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

1. 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

1. 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('新密码') where user='root';

flush privileges;

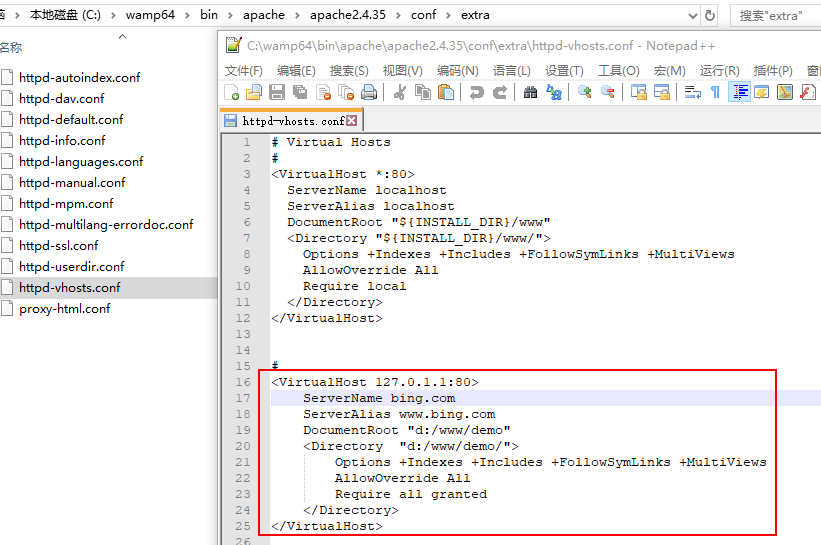
## 修改wampserver的www路径

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';

## 添加virtual host

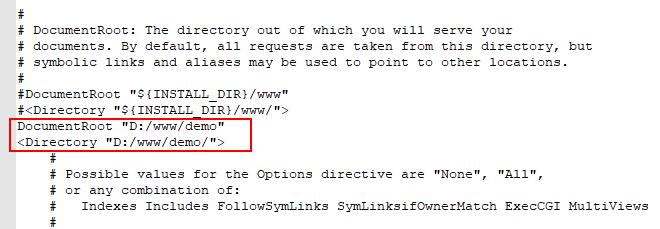
不同虚拟主机对应不同网站实例，方便切换调试。但其他主机无法直接访问虚拟主机。



## Apache允许其他主机访问项目目录

文件：http.conf

查找“DocumentRoot”, 修改后面的路径，改为指定项目目录。



文件：http.conf

找到想要被访问的文件对应的目录位置，将Require local 修改为 Require all granted



## 修改phpadmin登录方式

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



# 

# PHP Composer

## 简介

### 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

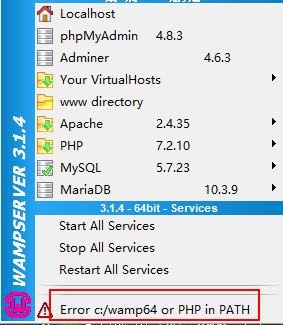
PSR-4 (Improved Autoloading) 自动加载优化标准

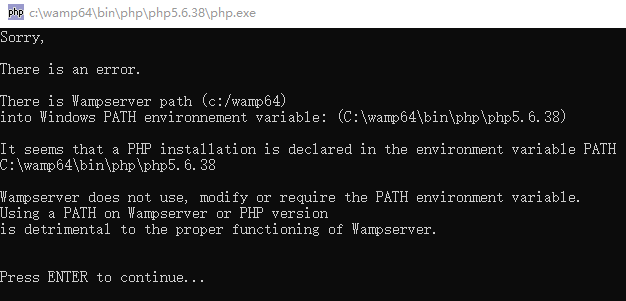
## 安装

Windows系统，下载运行 composer.exe，自动添加环境变量。

### 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：



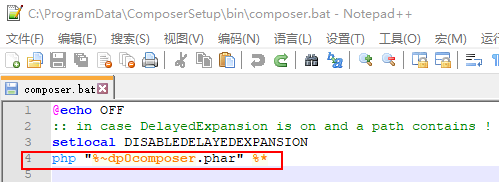


规避方法：

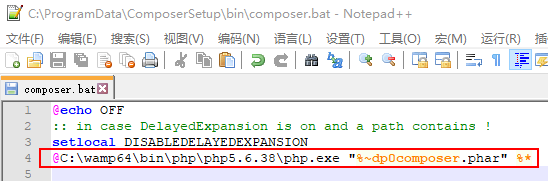
修改composer.bat来解决，直接修改C:\ProgramData\ComposerSetup\bin\composer.bat文件。

@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

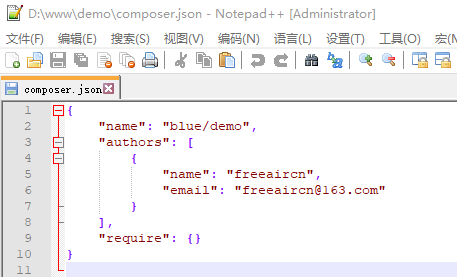
composer config repo.packagist composer <https://packagist.phpcomposer.com>

## 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：



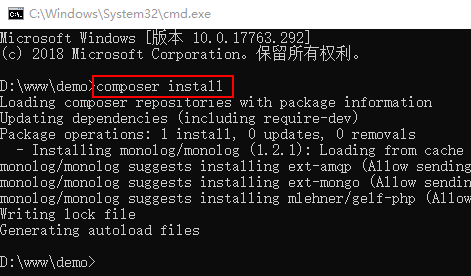
或者使用命令行，单独执行require和update

composer require monolog/monolog: 1.0.0

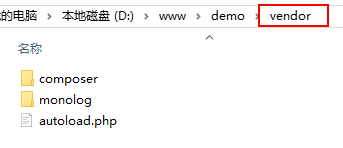
composer update monolog/monolog: 1.0.2

### 安装依赖包

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

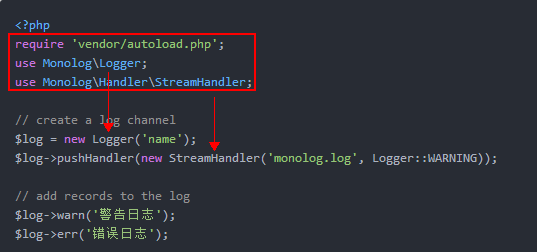


默认，在项目根目录下创建“vendor”文件夹，并将依赖包下载到vendor内，生成了autoload.php自动加载文件。



### 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。

## 项目适配



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

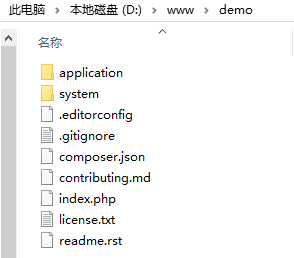


# 

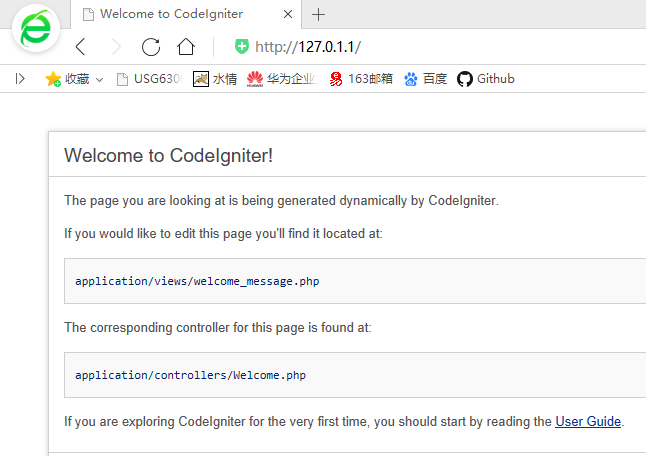
# CodeIgniter和admin框架适配

## CI初始配置

下载源文件后，解压至wamp虚拟主机项目的根目录下。



启动wamp服务，测试虚拟主机地址，默认主页显示成功。



## CI去掉默认url中的index.php

1.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。

搜索 AllowOverride None（配置文件中有多处），看注释信息，将相关.htaccess的该行信息改为:

AllowOverride All

2.在CI的根目录下，即在index.php，system的同级目录下，建立.htaccess，直接建立该文件名的不会成功，可以先建立记事本文件，另存为该名的文件即可。内容如下(CI手册上也有介绍):

RewriteEngine on

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

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

如果文件不是在www的根目录下，例如我的是：

http://localhost/ci\_demo\_1/index.php/

第三行需要改写为

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

另外，我的index.php的同级目录下还有assets文件夹，这些需要过滤除去，第二行需要改写为：

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

3.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。

移除 URL 中的 index.php

默认情况，你的 URL 中会包含 index.php 文件:

example.com/index.php/news/article/my\_article

如果你的 Apache 服务器启用了 mod\_rewrite ，你可以简单的通过一个 .htaccess 文件再加上一些简单的规则就可以移除 index.php 了。下面是这个文件的一个例子， 其中使用了 "否定条件" 来排除某些不需要重定向的项目：

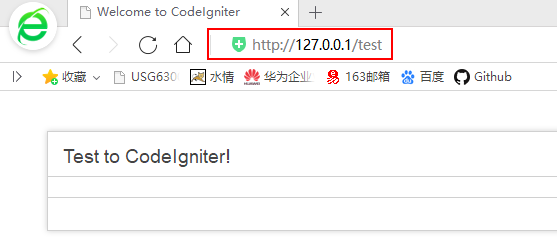
RewriteEngine On

RewriteCond %{REQUEST\_FILENAME} !-f

RewriteCond %{REQUEST\_FILENAME} !-d

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

在上面的例子中，除已存在的目录和文件，其他的 HTTP 请求都会经过你的 index.php 文件。



## 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iview admin

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。



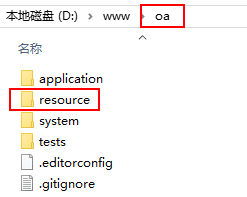
1. 验证，通过CI打开iview admin默认主页



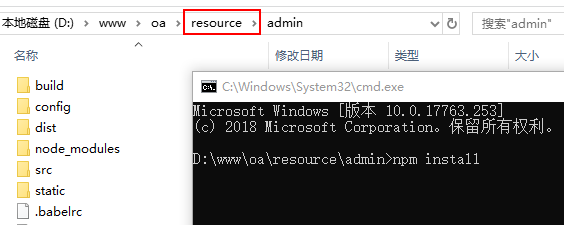
## CI项目引入element admin

### CI目录添加element admin

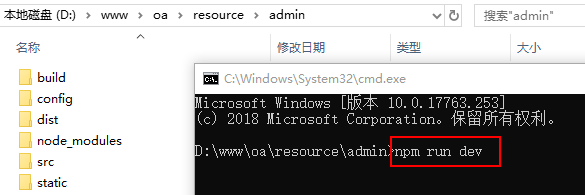
1. CI根目录下新建resource文件夹，存放资源文件。



1. 将GitHub下载的element admin解压后的文件拷贝到resource文件夹。首次拷贝，需要npm install安装项目依赖包。

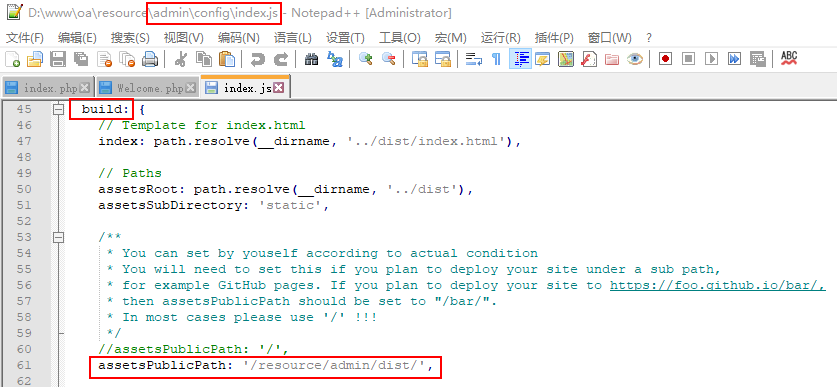


1. 验证element admin：



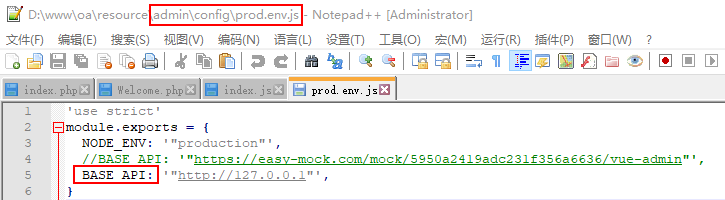
### CI引入element admin

1. 修改element admin的config文件：

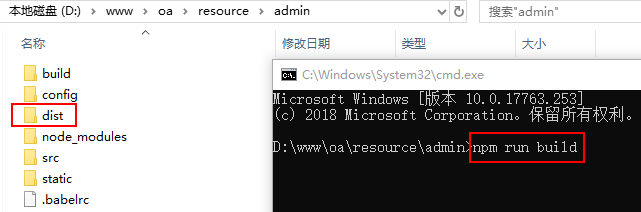


1. 修改pro的config文件：

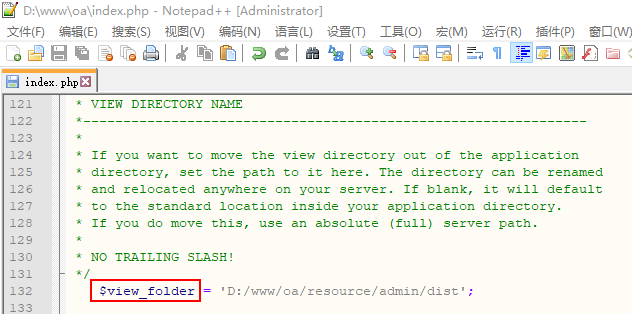




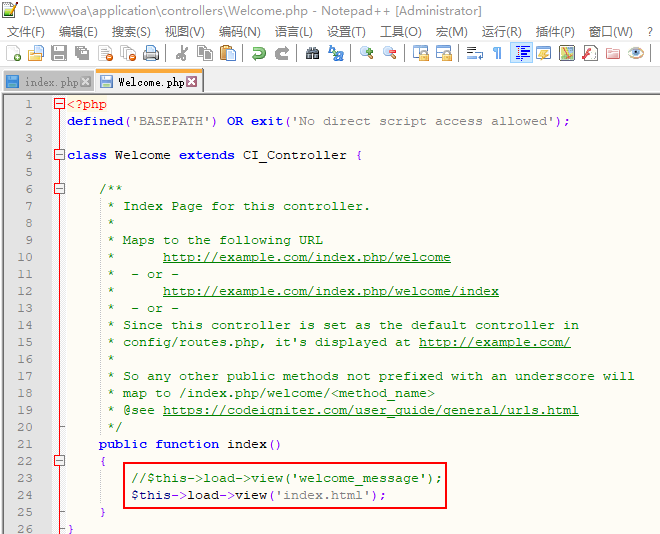
1. 生产element admin



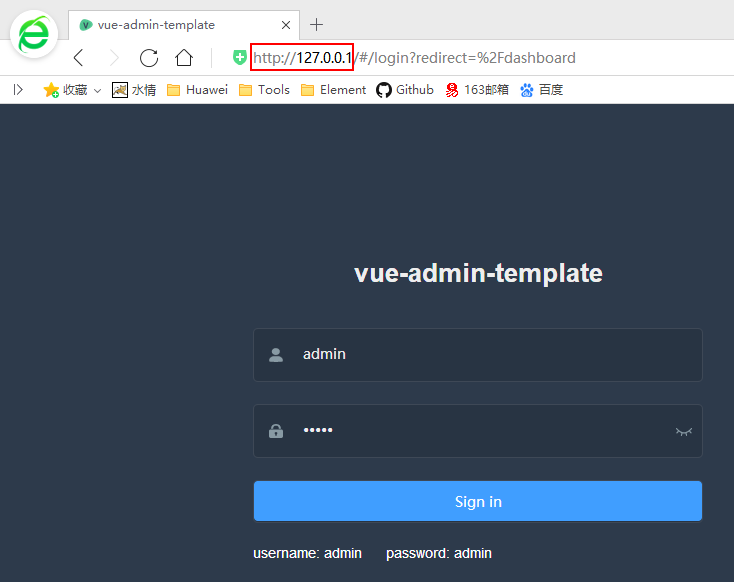
1. CI index.php文件，view folder填写element admin的dist绝对路径。



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



1. 验证，通过CI打开默认主页



# 附录

## A