<https://www.jianshu.com/p/155b3f18abae>

**在github中使用Travis CI**

[](https://www.jianshu.com/u/f221deffdd5b)

[饶曉文](https://www.jianshu.com/u/f221deffdd5b) 关注

2017.01.02 18:43\* 字数 1549 阅读 259评论 0喜欢 3



head

持续集成：Continuous Integration，简称CI，意思是，在一个项目中，任何人对代码库的任何改动，都会触发CI服务器自动对项目进行构建，自动运行测试，甚至自动部署到测试环境。这样做的好处就是，随时发现问题，随时修复。因为修复问题的成本随着时间的推移而增长，越早发现，修复成本越低。

[Travis CI](https://link.jianshu.com/?t=https://travis-ci.org)是在线托管的CI服务，用Travis来进行持续集成，不需要自己搭服务器，在网页上点几下就好，用起来更方便。最重要的是，它对开源项目是免费的。

**关联github和TravisCI**

[github](https://link.jianshu.com/?t=https://github.com)+[TravisCI](https://link.jianshu.com/?t=https://travis-ci.org)是一对好基友，现在很多开源项目都使用了Travis CI服务，你通常可以在相关项目的readme.md中的一开头看到文章开头的图片：build pass。这是个实时显示的图标，表示编译通过。

是不是看起来非常酷？要怎么才能做到呢？请按照以下步骤：

1. 用github账号登录并授权Travis CI
2. 在右上角你的账户名点击进入 account，在Repositories tab页点击Sync now同步你的github项目
3. 选中项目将默认的off改变为on开启项目的持续集成
4. 在你项目的根目录建立一个.travis.yml文件，按照规则编写。
5. 修改readme.md增加图标，markdown语法这样写:
6. [![Build Status](https://travis-ci.org/<github\_id>/<repo\_name>.svg?branch=dev)](https://travis-ci.org/<github\_id>/<repo\_name>)

github\_id:就是你的github登录名，repo\_name就是你的仓库名字。

1. 进行git push操作就可以触发Travis CI的进行集成构建。

**.travis.yml规则**

首先官方的doc在这里：[docs](https://link.jianshu.com/?t=https://docs.travis-ci.com" \t "_blank)。

简单的说，Travis CI并不帮你规定如何构建如何检查，所有规则与工具应该是你提供的或者能找到的，它只帮你提供了一个满足你项目构建检查的一个环境，包括操作系统类型等麻烦的事情。

一个构建的流程，在.travis.yml中体现下面

* [apt addons](https://link.jianshu.com/?t=https://docs.travis-ci.com/user/installing-dependencies/#Installing-Packages-with-the-APT-Addon)，简单地说就是通过apt包管理器来安装一些依赖包(可选的安装)
* [cache](https://link.jianshu.com/?t=https://docs.travis-ci.com/user/caching)(可选的安装):缓存一些不常改变的东西，加快编译速度
* before\_install,在install标签之前的动作
* install，安装依赖或者工具
* before\_script：script执行之前，可选
* script:这个标签下的动作是由bash来解析的，这里放构建的主要步骤
* before\_cache(可选的安装)，用于清除cache
* before\_deploy：部署之前，可选
* deploy：部署，可选
* after\_deploy：部署，可选
* after\_script：script执行之后，可选
* after\_success：构建成功之后的动作
* after\_failure：构建失败之后的动作

下面是install标签，而且有两个步骤，多个步骤就用-开始。

install:

- bundle install --path vendor/bundle

- npm install

如果before\_install,install或者before\_script返回非0，那么构建将立即停止并提示错误。

而script则会继续执行。

after\_success, after\_failure, after\_script这三个标签对构建的结果没有任何影响。

指定repo分支：

# blocklist

branches:

except:

- legacy

- experimental

# safelist

branches:

only:

- master

- stable

还可以用正则表达式来匹配：

branches:

only:

- master

- /^deploy-.\*$/

一般来说，复杂的构建过程，不应该直接写在.travis.yml中，而是应该在仓库里面写好构建脚本，然后再从.travis.yml中调用。

这么说，你就知道了，TravisCI只是提供个方便的构建环境而已。

**安装依赖**

apt包管理器：

before\_install:

- sudo apt-get -qq update

- sudo apt-get install -y libxml2-dev

第三方ppa：

before\_install:

- sudo add-apt-repository ppa:ubuntu-toolchain-r/test -y

- sudo apt-get update -q

- sudo apt-get install gcc-4.8 -y

没有apt也没有ppa的:

before\_install:

- wget http://pngquant.org/pngquant\_1.7.1-1\_i386.deb

- sudo dpkg -i pngquant\_1.7.1-1\_i386.deb

**Apt Addon**

addons:

apt:

sources:

- deadsnakes

- sourceline: 'ppa:ubuntu-toolchain-r/test'

- sourceline: 'deb https://packagecloud.io/chef/stable/ubuntu/precise main'

key\_url: 'https://packagecloud.io/gpg.key'

其中sourceline将后面获取的文本添加到/etc/apt/sources.list

key\_url后面的是GPG keys.

addons:

apt:

packages:

- cmake

- time

addons:

apt:

sources:

- lucid

packages:

- libcxsparse3.1.2

macOS

before\_install:

- brew update

- brew install beanstalk

缓存那些不经常变的东西，加速构建过程：

cache:

directories:

- .autoconf

- $HOME/.m2

[cacheing](https://link.jianshu.com/?t=https://docs.travis-ci.com/user/caching/)

**构建的环境变量**

[默认的环境变量](https://link.jianshu.com/?t=https://docs.travis-ci.com/user/environment-variables/#Default-Environment-Variables)

典型的设置，这样的设置将触发两次构建，分别以这些变量的不同值来构建

env:

- FOO=foo BAR=bar

- FOO=bar BAR=foo

**多操作系统**

os:

- linux

- osx

before\_install:

- if [[ "$TRAVIS\_OS\_NAME" == "osx" ]]; then brew update ; fi

- if [[ "$TRAVIS\_OS\_NAME" == "osx" ]]; then brew install graphviz; fi

TRAVIS\_OS\_NAME表示操作系统的类型

允许某个操作系统上的失败:

matrix:

allow\_failures:

- os: osx

matrix:

include:

- os: linux

dist: trusty

sudo: required

- os: osx

osx\_image: xcode7.2

**矩阵**

matrix是一个这样的标签，用于处理矩阵编译的情况，比如说：

rvm:

- 1.9.3

- 2.0.0

- 2.2

- ruby-head

- jruby

- rbx-2

- ree

gemfile:

- gemfiles/Gemfile.rails-2.3.x

- gemfiles/Gemfile.rails-3.0.x

- gemfiles/Gemfile.rails-3.1.x

- gemfiles/Gemfile.rails-edge

env:

- ISOLATED=true

- ISOLATED=false

上面的定义将形成7 \* 4 \* 2=56种不同的构建，这就是所谓的矩阵。

matrix:

exclude:

- rvm: 1.9.3

gemfile: gemfiles/Gemfile.rails-2.3.x

env: ISOLATED=true

- rvm: jruby

gemfile: gemfiles/Gemfile.rails-2.3.x

env: ISOLATED=true

上面的就实际上排除了56种情况中的两种，现在剩下54种构建的情况。

除了exclude还有include，可以特定加入一种特殊的情况，这样做的好处是再不让矩阵变大的情况下，拥有更大的灵活性。

matrix:

include:

- rvm: ruby-head

gemfile: gemfiles/Gemfile.rails-3.2.x

env: ISOLATED=false

注意ruby-head是上面矩阵里面没有的。

**通知**

notifications:

email:

- one@example.com

- other@example.com

全部关闭：

notifications:

email: false

条件控制:

notifications:

email:

recipients:

- one@example.com

- other@example.com

on\_success: [always|never|change] # default: change

on\_failure: [always|never|change] # default: always

always和never表示总是发送或者从不发送，change表示状态改变的时候发送.

通过irc来通知:

notifications:

irc:

channels:

- "chat.freenode.net#my-channel"

- "chat.freenode.net#some-other-channel"

on\_success: [always|never|change] # default: always

on\_failure: [always|never|change] # default: always

自定义消息类型:

notifications:

irc:

channels:

- "chat.freenode.net#my-channel"

- "chat.freenode.net#some-other-channel"

template:

- "%{repository} (%{commit}) : %{message} %{foo} "

- "Build details: %{build\_url}"

详情看[IRC-notification](https://link.jianshu.com/?t=https://docs.travis-ci.com/user/notifications/" \l "IRC-notification" \t "_blank)

**在vim项目中使用travisCI**

楼主是个vim爱好者，所以用这个来介绍😝

仓库地址:[vimagit](https://link.jianshu.com/?t=https://github.com/jreybert/vimagit" \t "_blank)

language: python

branches:

only:

- master

- next

- /^dev\/.\*$/

os:

- linux

# - osx

env:

- VIM\_VERSION=normal

- VIM\_VERSION=last

- VIM\_VERSION=neovim

- VIM\_VERSION=macvim

matrix:

exclude:

- os: osx

env: VIM\_VERSION=last

# neovim build fails with homebrew

# Error: undefined method `desc' for Neovim:Class

- os: osx

env: VIM\_VERSION=neovim

- os: linux

env: VIM\_VERSION=macvim

install:

- if [ "$TRAVIS\_OS\_NAME" = 'osx' ]; then

if [ "$VIM\_VERSION" = 'neovim' ]; then

brew tap neovim/neovim &&

brew tap --repair &&

brew install --HEAD neovim;

elif [ "$VIM\_VERSION" = 'macvim' ]; then

brew install macvim;

elif [ "$VIM\_VERSION" = 'normal' ]; then

echo "use normal vim";

else

echo "VIM\_VERSION is not set";

exit 1;

fi

elif [ "$TRAVIS\_OS\_NAME" = 'linux' ]; then

if [ "$VIM\_VERSION" = 'last' ]; then

sudo add-apt-repository -y ppa:fcwu-tw/ppa &&

sudo apt-get -qq update &&

sudo apt-get -qq -f install &&

sudo apt-get -qq install vim;

elif [ "$VIM\_VERSION" = 'neovim' ]; then

sudo add-apt-repository -y ppa:neovim-ppa/unstable &&

sudo apt-get -qq update &&

sudo apt-get -qq -f install &&

sudo apt-get -qq install neovim;

elif [ "$VIM\_VERSION" = 'normal' ]; then

echo "use normal vim";

else

echo "VIM\_VERSION is not set";

exit 1;

fi

fi

before\_script:

- git clone https://github.com/jreybert/djooks

- git clone https://github.com/jreybert/vader.vim

script:

- ./test/run.sh . vader.vim djooks $VIM\_VERSION

after\_success:

- ./test/merge.sh

一个标签一个标签说：

第一个branches这里选择了master,next和dev开头的分支来编译

os选择了linux

env定义了VIM\_VSERION用于下面的install标签，也表示以此将有四种不同的编译。

matrix:排除了一些构建的情况：osx下不要编译VIM\_VERSION=last，neovim，macvim

install标签，根据VIM\_VERSION变量进行依赖的安装

before\_script:clone了两个工具

script:执行了本仓库下的test/run.sh脚本

after\_success：如果成功，执行本仓库下的./test/merge.sh脚本，查看其内容是将非主分支的合并到主分支中

本人并不使用示例中的vader.vim工具，而是采用更强大的：[Kuniwak/vint](https://link.jianshu.com/?t=https://github.com/Kuniwak/vint)  
和[syngan/vim-vimlint](https://link.jianshu.com/?t=https://github.com/syngan/vim-vimlint)

具体使用与按照看连接，并非本文内容。