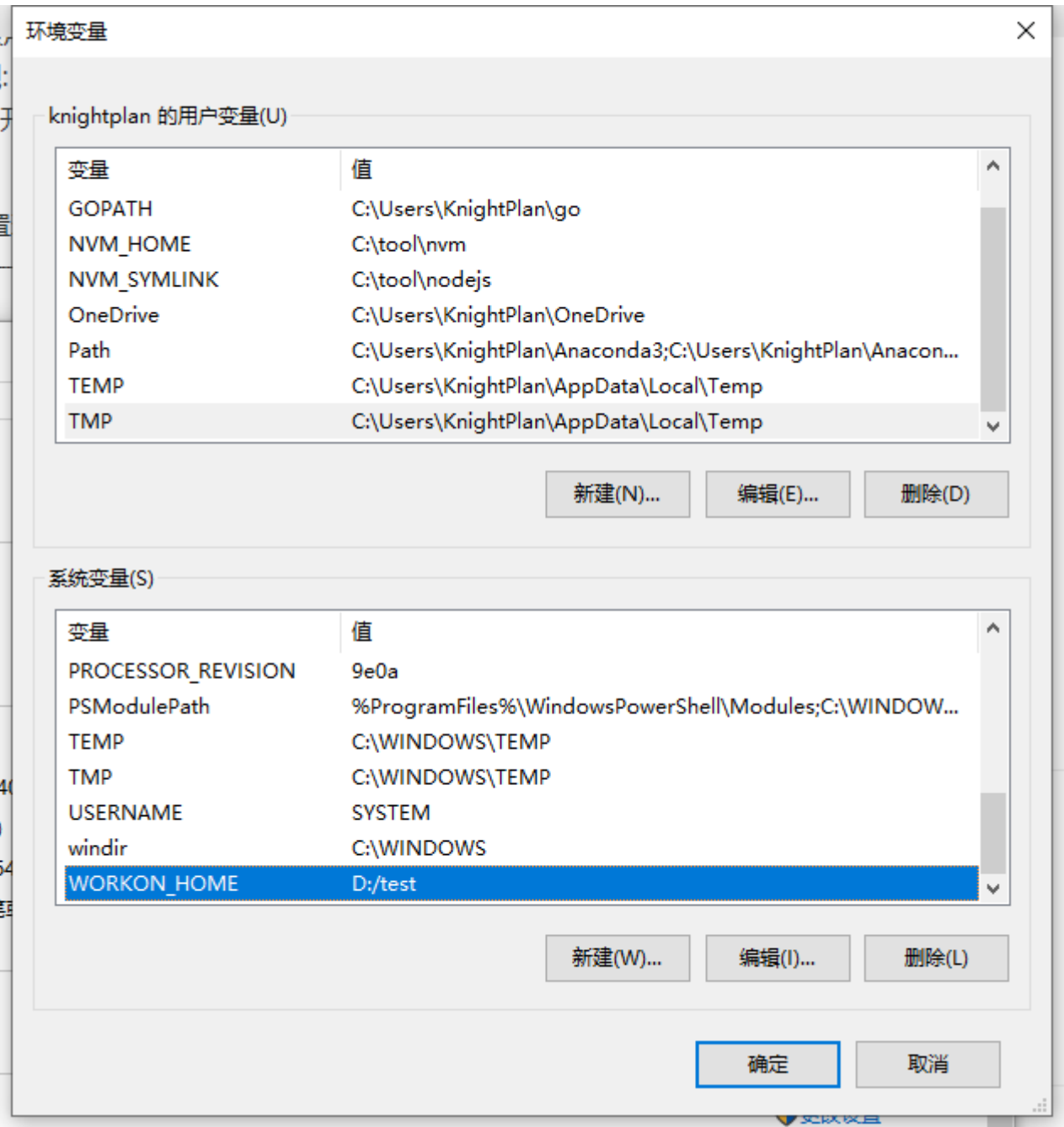


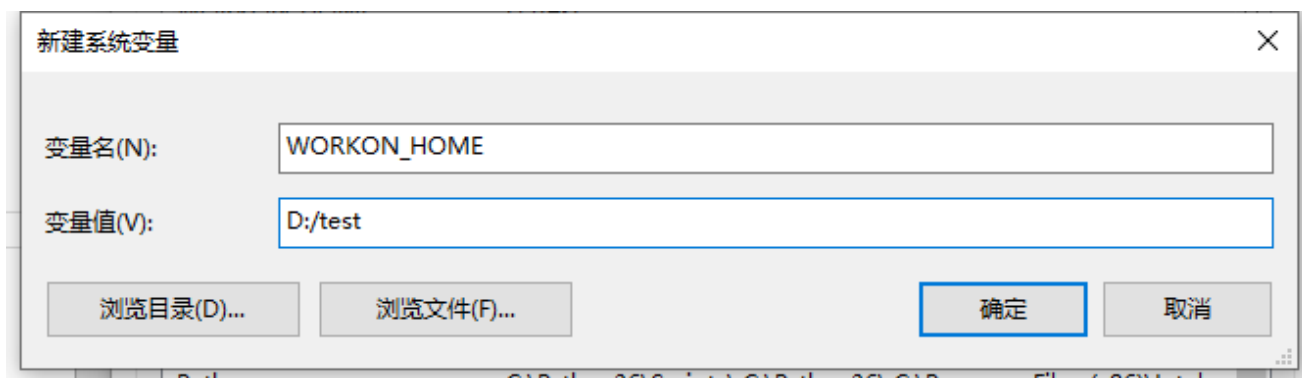
# 昨天问题总结

1. 安装了虚拟环境以后,无法使用deactivate退出环境,并且使用pip list输出的当前环境模块是全局环境中的.

**问题出现的原因:** 当前系统中,登录用户的家目录是中文的! **防范类似的问题出现:** 安装的软件\项目开发使用到的路径\开发相关的目录不要涉及到中文 **解决方案:**

1. 在当前电脑中,配置环境变量中添加一个配置虚拟环境存储目录的变量





2. 数据创建的账号无法登陆,原来的root账号没有问题,创建账号的sql语句没有问题.

#### 错误发生的原因:

在安装mysql的时候,没有进行初始化[没有删除匿名用户]

#### 防范出现这个问题:

以后安装了mysql以后,进入到mysql的控制台,找到mysql数据库的user表,把user=""的用户删除

```
delete from mysql.user where user=''
```

注意,如果在删除匿名用户之前已经创建的用户,这些用户是无法登陆,这些用户也需要删除  
删除了用户以后,还要从权限中移除上面账号的相关权限记录

```
drop user '无法登陆的用户名';      # 匿名用户无需进行这项操作
```

```
flush privileges;                    # 刷新mysql的权限记录,保证上面更改立马刷新
```

#### 解决方案:

同上面的防范措施一致.

## 如何多人协同开发同一个项目?

使用代码版本控制[version control]软件,

目前市面上比较流行的代码版本控制器有: git,svn,csv

## 1. 使用git管理代码版本

本项目使用git管理项目代码, 代码库放在gitee码云平台。(注意, 公司中通常放在gitlab私有服务器中)

### 1.1 Git 的诞生

作用：源代码管理

- 方便多人协同开发
- 方便版本控制

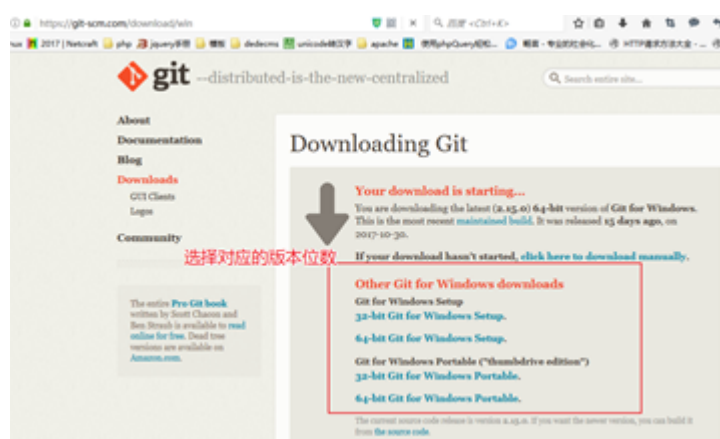
SVN 都是集中控制管理的，也就是有一个中央服务器，大家都把代码提交到中央服务器，而 git 是分布式的版本控制工具，也就是说没有中央服务器，每个节点的地位平等。

The diagram illustrates a peer-to-peer network topology. It features six nodes arranged in a circular pattern. Each node is represented by either a laptop or a desktop computer, all with blue screens. The nodes are interconnected by double-headed arrows, indicating that every node can communicate directly with every other node in the network. This setup allows for decentralized data storage and sharing, where each node can act as both a client and a server.

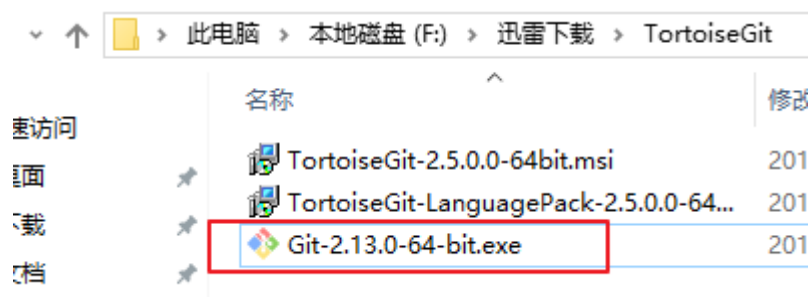
## Git的windows下安装

官网地址:

<https://git-scm.com/download>



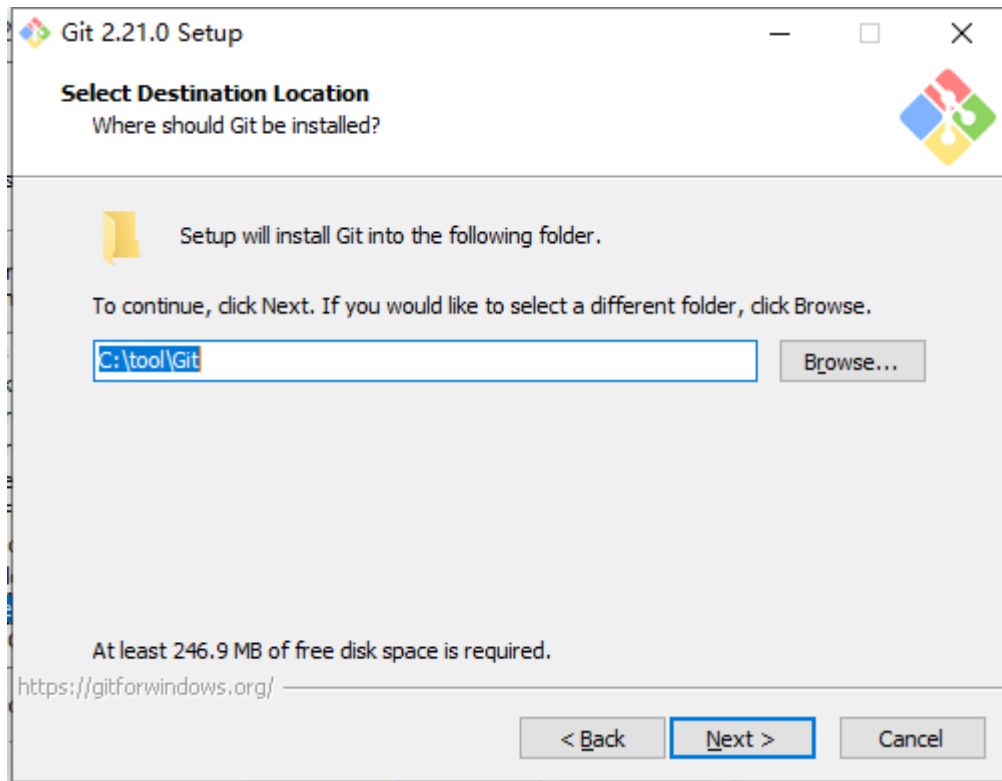
下载到本地磁盘

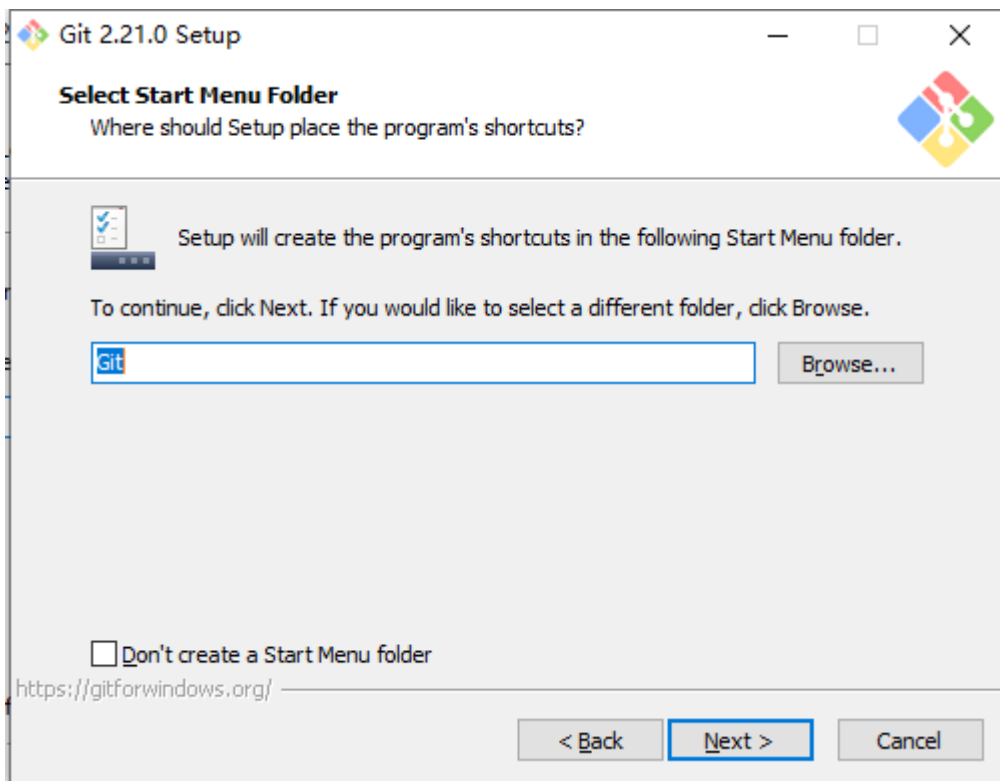
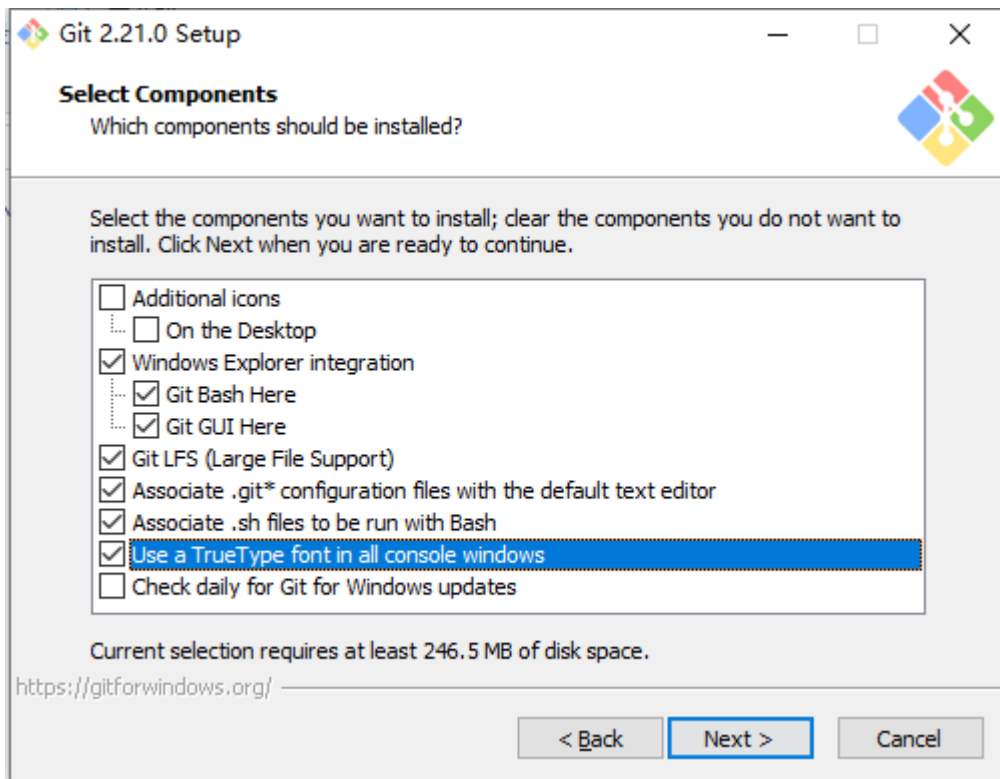


安装



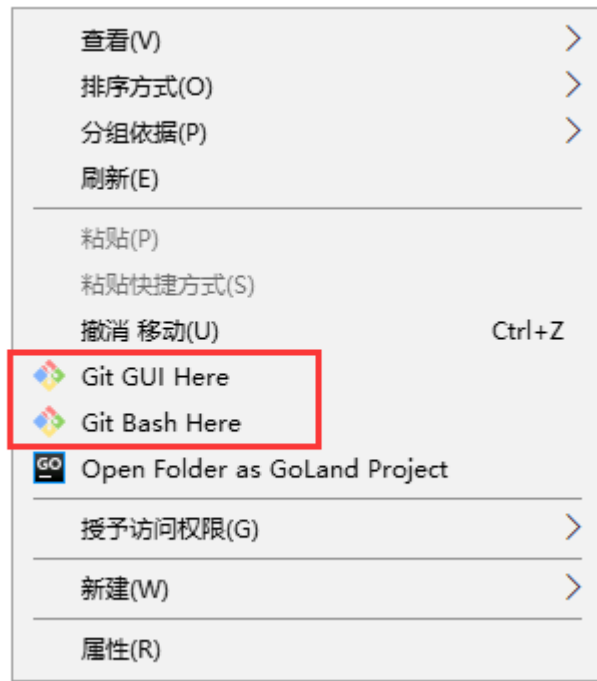
一路【next】就可以了



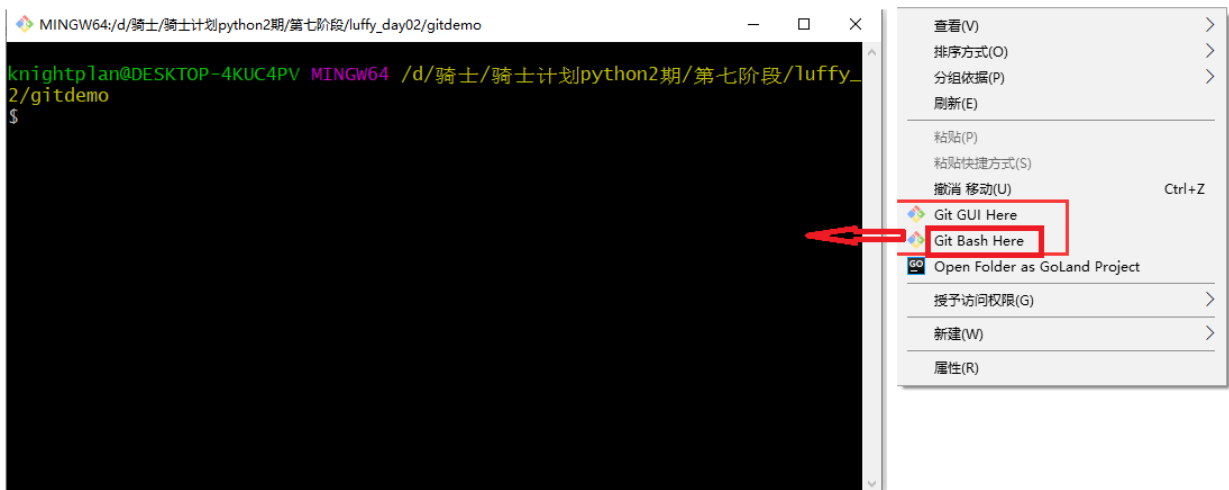


注意：openssl 一定选它

安装完成后，右击菜单栏，有如下菜单，表示安装完成



进入git bash选项



Git工作区、暂存区和版本库



# 1、工作区介绍

就是在你本要电脑磁盘上能看到的目录。

# 2、暂存区介绍

一般存放在【.git】目录下的index文件(.git/index) 中，所以我们把暂存区有时也叫作索引。

# 3、版本库介绍

工作区有一个隐藏目录.git，这个不算工作区，而是Git的版本库。git中的head/master是分支，是版本库。

## git服务器本地搭建

cd进入到自己希望存储代码的目录路径，并创建本地仓库.git  
新创建的本地仓库.git是个空仓库

```
cd 目录路径
git init
```

创建仓库

```
knighplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python2期/第七阶段/luffy_
day02/gitdemo
$ git init
Initialized empty Git repository in D:/骑士/骑士计划python2期/第七阶段/luffy_
_day02/gitdemo/.git/
knighplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python2期/第七阶段/luffy_
day02/gitdemo (master)
```

表示创建了一个空的代码库，同时显示当前代码库在系统中的位置，

## 配置用户名和邮箱

```
git config --global user.name 'lisi'
git config --global user.email 'lisi@163.com'
```



```

xiaowu@DESKTOP-451HRKT MINGW64 /f/www/class/zhan02/git_2 (master)
$ git config --global user.name 'lisi';

xiaowu@DESKTOP-451HRKT MINGW64 /f/www/class/zhan02/git_2 (master)
$

xiaowu@DESKTOP-451HRKT MINGW64 /f/www/class/zhan02/git_2 (master)
$ git config --global user.email 'lisi@126.com';

xiaowu@DESKTOP-451HRKT MINGW64 /f/www/class/zhan02/git_2 (master)
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
rebase.autosquash=true
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.required=true
filter.lfs.process=git-lfs filter-process
credential.helper=manager
user.name=lisi
user.email=lisi@126.com
core.repositoryformatversion=0
core.filemode=false

```

## 查看仓库状态

```
git status
```

```
git status -s 简约显示
```

- 红色表示新建文件或者新修改的文件,都在工作区.
- 绿色表示文件在暂存区
- 新建的 `login.py` 文件在工作区, 需要添加到暂存区并提交到仓库区

```

xiaowu@DESKTOP-451HRKT MINGW64 /f/www/class/zhan02/git_2 (master)
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)

```

上图表示: 暂时没有新文件需要提交到暂存区

## 添加文件到暂存区

```
# 添加项目中所有文件
git add .
或者
# 添加指定文件
git add login.py
```

例如：创建3个文件，并查看状态.

## 提交到版本库

```
git commit -am "版本描述"
```

## 手动删除文件

手动操作删除 或者 在命令行下 使用 `rm 文件名` 删除 都是表示在工作区删除。  
对于这种删除,如果还原,则可以使用 `git checkout 文件名`

```
knightplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python[redacted]/luffy_
day02/gitdemo (master)
$ rm 1.py

knightplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python[redacted]/luffy_
day02/gitdemo (master)
$ git checkout
D      1.py

knightplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python[redacted]/luffy_
day02/gitdemo (master)
$ git checkout 1.py
Updated 1 path from the index
```

## 版本删除

如果使用 `git rm 文件名`，这种操作属于暂存区删除,这种删除无法直接`git checkout 文件名` 来还原。  
如果直接执行`git checkout` 命令,则报错如下:

```
knightplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python[redacted]/luffy_
day02/gitdemo (master)
$ git checkout 1.py
error: pathspec '1.py' did not match any file(s) known to git
```

如果要还原在暂存区中删除的文件,必须先执行 `git reset head`

```
knighiplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python1期/第七阶段/luffy_
day02/gitdemo (master)
$ git reset head
Unstaged changes after reset:
D      1.py

knighiplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划python1期/第七阶段/luffy_
day02/gitdemo (master)
$ git checkout 1.py
Updated 1 path from the index
```

## 查看历史版本[查看日志]

```
git log    或者    git reflog
```

过滤查看日志

```
git log -p
```

退出按【q】键

ctrl+f 向下分页

ctrl+b 向上分页

显示指定日期之后的日志 `git log --after '2018-11-6'`

显示指定日期之前的日志 `git log --before '2018-11-6'`

指定显示指定开发者的日志 `git log --author 'lisi'`

## 回退版本

- 方案一:

- `HEAD` 表示当前最新版本
- `HEAD^` 表示当前最新版本的前一个版本
- `HEAD^^` 表示当前最新版本的前两个版本, 以此类推...
- `HEAD~1` 表示当前最新版本的前一个版本
- `HEAD~10` 表示当前最新版本的前10个版本, 以此类推...

```
git reset --hard HEAD^
```

## 方案二: 当版本非常多时可选择的方案

- 通过每个版本的版本号回退到指定版本

```
git reset --hard 版本号
```

取消暂存

```
git reset head
git reset <file> : 从暂存区恢复到工作文件
git reset -- : 从暂存区恢复到工作文件
```

## 查看文件状态

**针对与文件所处的不同分区，文件所处的状态：**

- (1)未追踪, 文件第一次出现在工作区, 版本库还没有存储该文件的状态
- (2)已追踪, 只要第一次,git add了文件, 文件就是已追踪
- (3)未修改, 文件在工作区未被编辑
- (4)已修改, 文件在工作区被修改
- (5)未暂存, 文件已修改, 但是没有add到暂存区
- (6)已暂存, 已经将修改的文件add到暂存区
- (7)未提交, 已暂存的文件, 没有commit提交. 处于暂存区
- (8)已提交, 提交到版本库的文件修改,只有commit以后才会有仓库的版本号生成

注意：

公司一般使用git管理项目,往往会搭建一个gitlab自己内部管理代码,也有公司选择使用码云的企业版仓库来管理

使用git开发项目时,有时候不一定通过https协议提交代码的。也有的公司是通过ssh协议提交,此时需要生成ssh公钥和提交公钥给仓库。[码云这些官网都会有详细的提示说明]

生成SSH公钥【必须安装git bash才可以使用这个命令，而且还要把git bash添加到系统变量里面】

```
ssh-keygen -t rsa -C "lisi@163.com"
```

参考：<https://gitee.com/help/articles/4180>

## 2. 在git平台创建工程

- 1) 创建私有项目库

# 新建仓库

仓库名称

mydjango

一个项目一个仓库, 仓库名称往往就是项目目录名

归属

mooluo

路径

https://gitee.com/mooluo/mydjango

仓库介绍 非必填

一个私有项目

是否开源

☒ 私有

☐ 公开

选择语言

Python

添加 .gitignore

请选择 .gitignore 模板

添加开源许可证

请选择开源许可证

☐ 使用Readme文件初始化这个仓库

☐ 使用Issue模板文件初始化这个仓库

☐ 使用Pull Request模板文件初始化这个仓库

导入已有仓库

创建

作为初学者, 这里我们不要勾选, 其实公司里创建项目也是空仓库

创建私有空仓库以后的界面:

[代码](#) [Issues 0](#) [Pull Requests 0](#) [附件 0](#) [Wiki 0](#) [DevOps](#) [服务](#) [管理](#)

快速设置—如果你知道该怎么操作，直接使用下面的地址

HTTPS

SSH

https://gitee.com/mooluo/mydjango.git

我们强烈建议所有的git仓库都有一个 `README`，`LICENSE`，`.gitignore` 文件

Git入门? 查看 [帮助](#)，[Visual Studio](#) / [TortoiseGit](#) / [Eclipse](#) / [Xcode](#) 下如何连接本站，[如何导入仓库](#)

简易的命令行入门教程:

Git 全局设置:

```
git config --global user.name "mooluo"
git config --global user.email "649641514@qq.com"
```

创建 git 仓库:

```
mkdir mydjango
cd mydjango
git init
touch README.md
git add README.md
git commit -m "first commit"
git remote add origin https://gitee.com/mooluo/mydjango.git
git push -u origin master
```

已有仓库?

```
cd existing_git_repo
git remote add origin https://gitee.com/mooluo/mydjango.git
git push -u origin master
```

## 2) 克隆项目到本地

`git clone` 仓库地址

注意，如果当前目录下出现git仓库同名目录时，会克隆失败。

```
knighplan@DESKTOP-4KUC4PV MINGW64 /d/骑士/骑士计划pytho
n2期/第七阶段
$ git clone https://gitee.com/mooluo/luffy.git
Cloning into 'luffy'...
warning: You appear to have cloned an empty repository.
```

## 3) 创建并切换分支到dev

|                                  |                      |
|----------------------------------|----------------------|
| <code># git branch dev</code>    | # 创建本地分支dev, dev是自定义 |
| <code># git checkout dev</code>  | # 切换本地分支代码           |
| <code>git checkout -b dev</code> | # 这里是上面两句代码的简写       |

git提交

```
git add 代码目录
git status
git commit -m '添加项目代码'
```

推送到远端

```
git push origin dev:dev
```

如果推送代码,出现以下提示: git pull ....,则表示当前本地的代码和线上的代码版本不同.

1. 把线上的代码执行以下命令,拉取到本地,进行同步  
git pull
2. 根据提示,移除多余的冲突的文件,也可以删除.  
完成这些步骤以后,再次add,commit,push即可.

## 3. 搭建前端项目

### 3.1 创建项目目录

```
cd 项目目录
vue init webpack lufei
```

例如,我要把项目保存在D:/moluo的根目录下,可以如下操作:

```
D:
cd /moluo
vue init webpack lufei
```

根据需要在生成项目时,我们选择对应的选项,效果:

```
D:\骑士\骑士计划pythor ----- 阶段 vue init webpack luffy_pc
? Project name luffy_pc
? Project description A Vue.js project
? Author moluo <123@qq.com>
? Vue build standalone
? Install vue-router? No
? Use ESLint to lint your code? No
? Set up unit tests No
? Setup e2e tests with Nightwatch? No
? Should we run `npm install` for you after the project has been created? (recommended) npm

vue-cli · Generated "luffy_pc".

# Installing project dependencies ...
#
[.....] \ fetchMetadata: sill install loadAllDepsIntoIdealTree
```

根据上面的提示，我们已经把vue项目构建好了，接下来我们可以在pycharm编辑器中把项目打开并根据上面黄色提示，运行测试服务器。

```
C:\WINDOWS\system32\cmd.exe
# =====
npm WARN deprecated browserslist@2.11.3: Browserslist 2 could fail on reading Browserslist >3.0 config used in other tools.
npm WARN deprecated bfj-node4@5.3.1: Switch to the `bfj` package for fixes and new features!
npm WARN deprecated browserslist@1.7.7: Browserslist 2 could fail on reading Browserslist >3.0 config used in other tools.
> uglifyjs-webpack-plugin@0.4.6 postinstall D:\骑士\骑士计划python2期\第七阶段\luffy_pc\node_modules\webpack\node_modules\uglifyjs-webpack-plugin
> node lib/post_install.js

npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN ajv-keywords@3.4.0 requires a peer of ajv@^6.9.1 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

added 1125 packages from 650 contributors and audited 10656 packages in 63.241s
found 8 vulnerabilities (1 low, 6 moderate, 1 high)
  run `npm audit fix` to fix them, or `npm audit` for details

# Project initialization finished!
# =====

To get started:

  cd luffy_pc
  npm run dev
```

打开项目已经，在pycharm的终端下运行vue项目，查看效果。

```
npm run dev
```

```
Terminal
+ D:\骑士\骑士计划python2期\第七阶段\luffy_pc>
x D:\骑士\骑士计划python2期\第七阶段\luffy_pc>npm run dev

> luffy_pc@1.0.0 dev D:\骑士\骑士计划python2期\第七阶段\luffy_pc
> webpack-dev-server --inline --progress --config build/webpack.dev.conf.js
```

效果：



Welcome to Your Vue.js App

Essential Links

[Core Docs](#) [Forum](#) [Community Chat](#) [Twitter](#)  
[Docs for This Template](#)

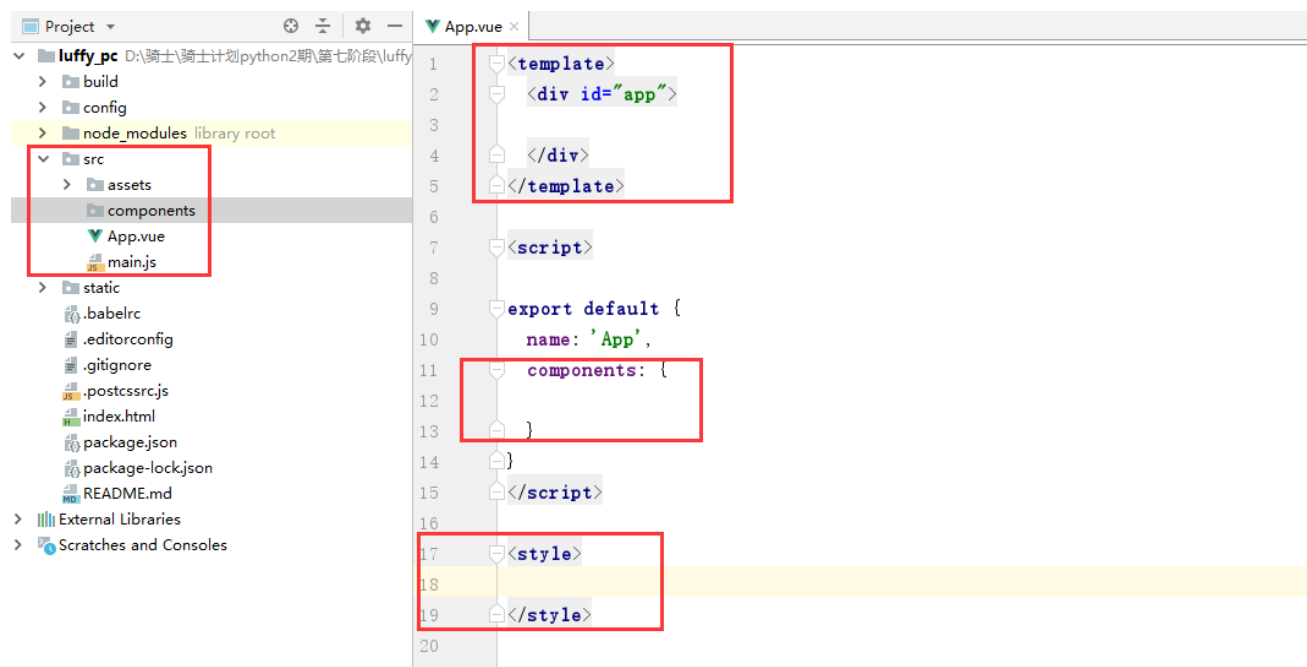


接下来，我们根据终端上效果显示的对应地址来访问项目(如果有多个vue项目在运行，8080端口被占据了，服务器会自动改端口，所以根据自己实际在操作中看到的地址来访问。)

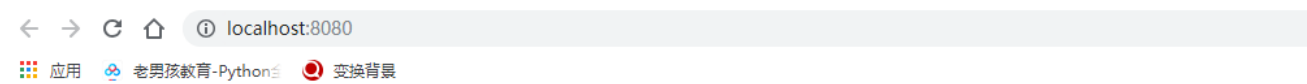
访问: <http://localhost:8080>

## 3.2 初始化前端项目

清除默认的HelloWorld组件和App.vue中的默认样式



接下来，我们可以查看效果了，一张白纸~



## 3.3 安装路由vue-router

### 3.3.1 下载路由组件

```
npm i vue-router -S
```

执行效果:

```
D:\骑士\骑士计划python2期\第七阶段\luffy_pc>
D:\骑士\骑士计划python2期\第七阶段\luffy_pc>npm i vue-router -S
npm WARN ajv-keywords@3.4.0 requires a peer of ajv@^6.9.1 but none is installed. You must install peer dependencies yourself.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

+ vue-router@3.0.2
added 1 package from 1 contributor and audited 10657 packages in 8.321s
found 8 vulnerabilities (1 low, 6 moderate, 1 high)
run `npm audit fix` to fix them, or `npm audit` for details
```

## 3.3.2 配置路由

### 3.3.2.1 初始化路由对象

在src目录下创建routers路由目录，在routers目录下创建index.js路由文件

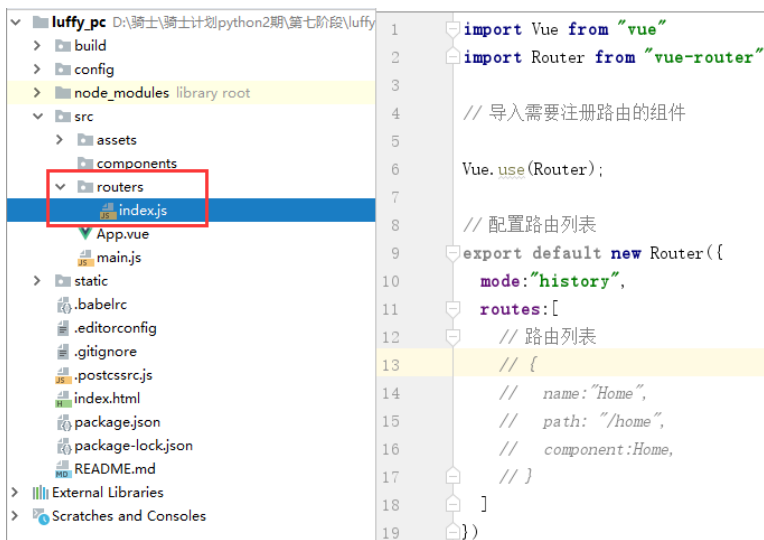
index.js路由文件中，编写初始化路由对象的代码。

```
import Vue from "vue"
import Router from "vue-router"

// 这里导入可以让用户访问的组件

Vue.use(Router);

export default new Router({
  // 设置路由模式为'history'，去掉默认的#
  mode: "history",
  routes: [
    // 路由列表
  ]
})
```



### 3.3.2.2 注册路由信息

打开main.js文件，把router对象注册到vue中。

代码：

```
// The Vue build version to load with the `import` command
// (runtime-only or standalone) has been set in webpack.base.conf with an alias.
import Vue from 'vue'
import App from './App'
import router from './routers/index';

Vue.config.productionTip = false

/* eslint-disable no-new */
new Vue({
  el: '#app',
  router,
  components: { App },
  template: '<App/>'
});
```

### 3.3.2.3 在视图中显示路由对应的内容

在App.vue组件中，添加显示路由对应的内容。



代码：

```
<template>
  <div id="app">
    <router-view/>
  </div>
</template>

<script>
export default {
  name: 'App',
  components: {
```

```
    }  
  }  
</script>  
  
<style>  
  
</style>
```

## 3.4 引入ElementUI

```
npm i element-ui -S
```

上面的命令等同于 `npm install element-ui --save`

执行命令效果：

```
D:\骑士\骑士计划python2期\第七阶段\luffy_pc>npm i element-ui -S  
npm WARN ajv-keywords@3.4.0 requires a peer of ajv@^6.9.1 but none is installed. You must install peer dependencies yourself.  
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.7 (node_modules\fsevents):  
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.7: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})  
  
+ element-ui@2.6.3  
added 6 packages from 6 contributors and audited 10667 packages in 10.677s  
found 8 vulnerabilities (1 low, 6 moderate, 1 high)  
  run `npm audit fix` to fix them, or `npm audit` for details  
  
D:\骑士\骑士计划python2期\第七阶段\luffy_pc>
```



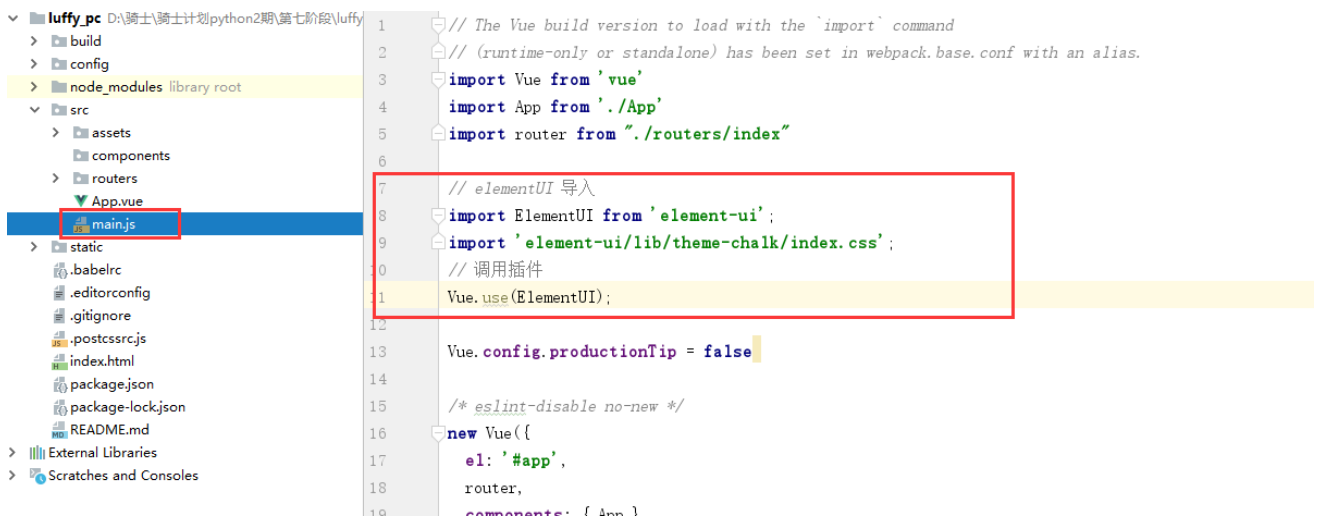
### 3.4.2 配置ElementUI到项目中

在main.js中导入ElementUI，并调用。

代码：

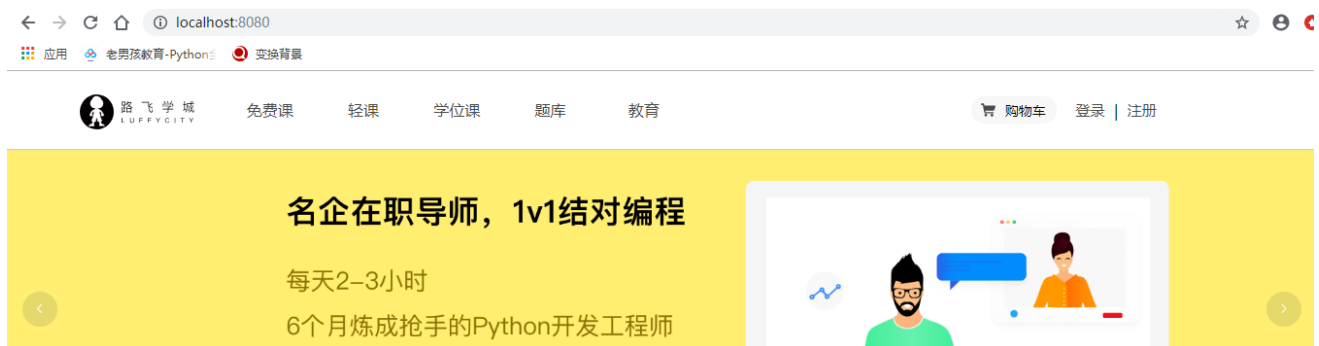
```
// elementUI 导入  
import ElementUI from 'element-ui';  
import 'element-ui/lib/theme-chalk/index.css';  
// 调用插件  
vue.use(ElementUI);
```

效果：



成功引入了ElementUI以后，接下来我们就可以开始进入前端页面开发，首先是首页。

接下来我们把之前完成的首页，直接拿过来使用[注意除了组件以外,还有静态文件也需要拿过来,包括App.vue里面的公共样式]，并运行项目。



## 4. 跨域CORS

我们现在为前端和后端分别设置两个不同的域名

window 系统: C:\Windows\System32\drivers\etc\hosts

linux/mac系统: /etc/hosts

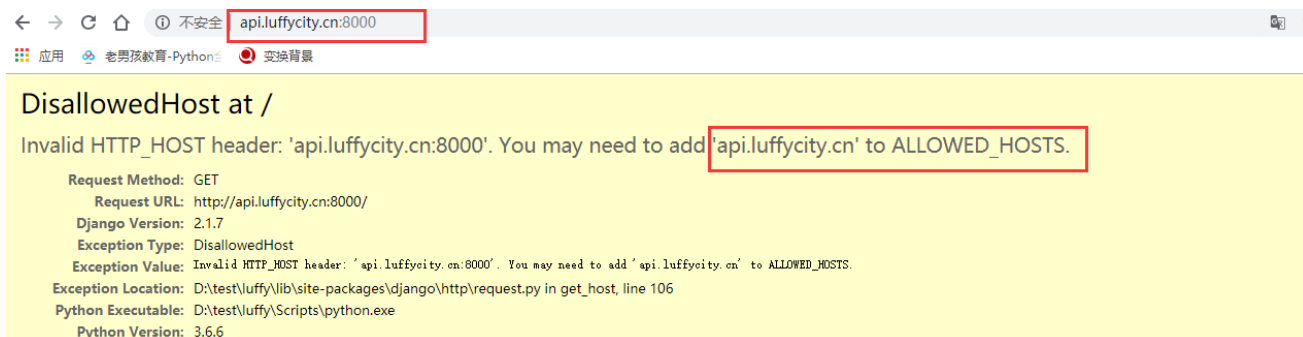
| 位置 | 域名               |
|----|------------------|
| 前端 | www.luffycity.cn |
| 后端 | api.luffycity.cn |

编辑 /etc/hosts 文件，可以设置本地域名

在文件中增加两条信息

```
127.0.0.1    api.luffycity.cn
127.0.0.1    www.luffycity.cn
```

通过浏览器访问drf项目,会出现以下错误信息



可以通过settings的ALLOWED\_HOSTS,设置允许访问

```
# 设置哪些客户端可以通过地址访问到后端
ALLOWED_HOSTS = [
    'api.luffycity.cn',
    'www.luffycity.cn',
    'localhost', # 实际开发的时候不会写上localhost和127.0.0.1的
    '127.0.0.1',
]
```

现在, 前端与后端分处不同的域名, 我们需要为后端添加跨域访问的支持。

否则前端无法使用axios无法请求后端提供的api数据

我们使用CORS来解决后端对跨域访问的支持。

使用django-cors-headers扩展

```
在 Response(headers={"Access-Control-Allow-Origin": '客户端地址/*'})
```

[文档](#)

安装

```
pip install django-cors-headers
```

添加应用

```
INSTALLED_APPS = (
    ...
    'corsheaders',
    ...
)
```

中间层设置【必须写在第一个位置】

```
MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    ...
]
```

添加白名单

```
# CORS组的配置信息
CORS_ORIGIN_WHITELIST = (
    '127.0.0.1:8080',
    'localhost:8080',
    'www.luffycity.cn:8080'
)
CORS_ALLOW_CREDENTIALS = True # 允许ajax跨域请求时携带cookie
```

完成了上面的步骤，我们就可以通过后端提供数据给前端使用ajax访问了。

## 5. redis

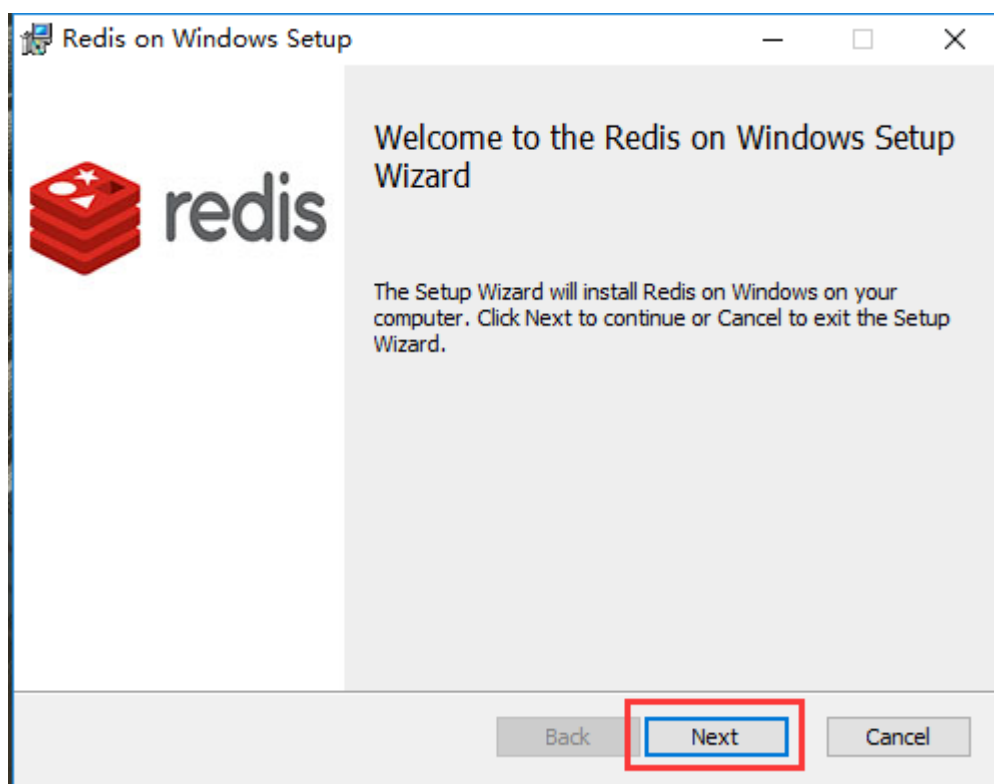
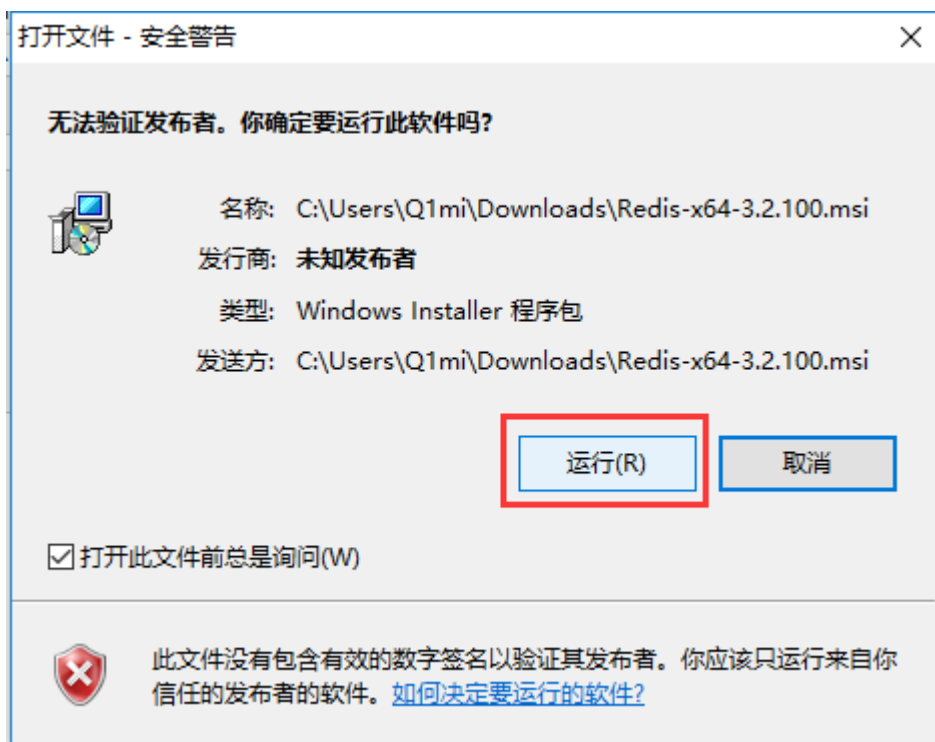
window系统的redis是微软团队根据官方的linux版本高仿的

官方原版: <https://redis.io/>

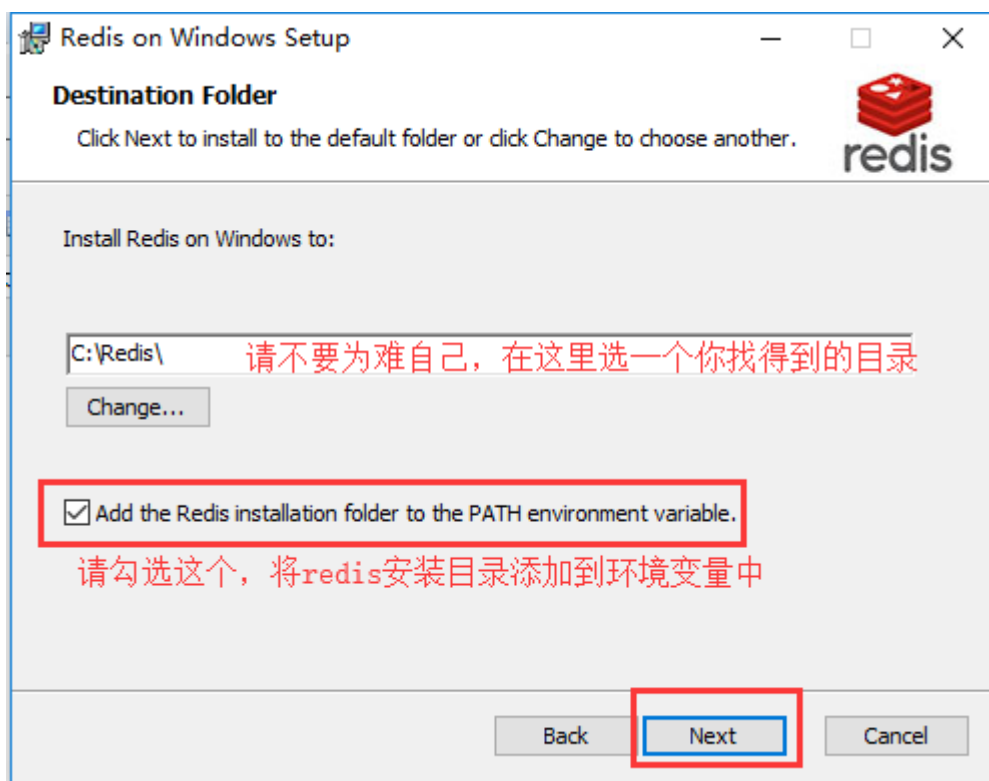
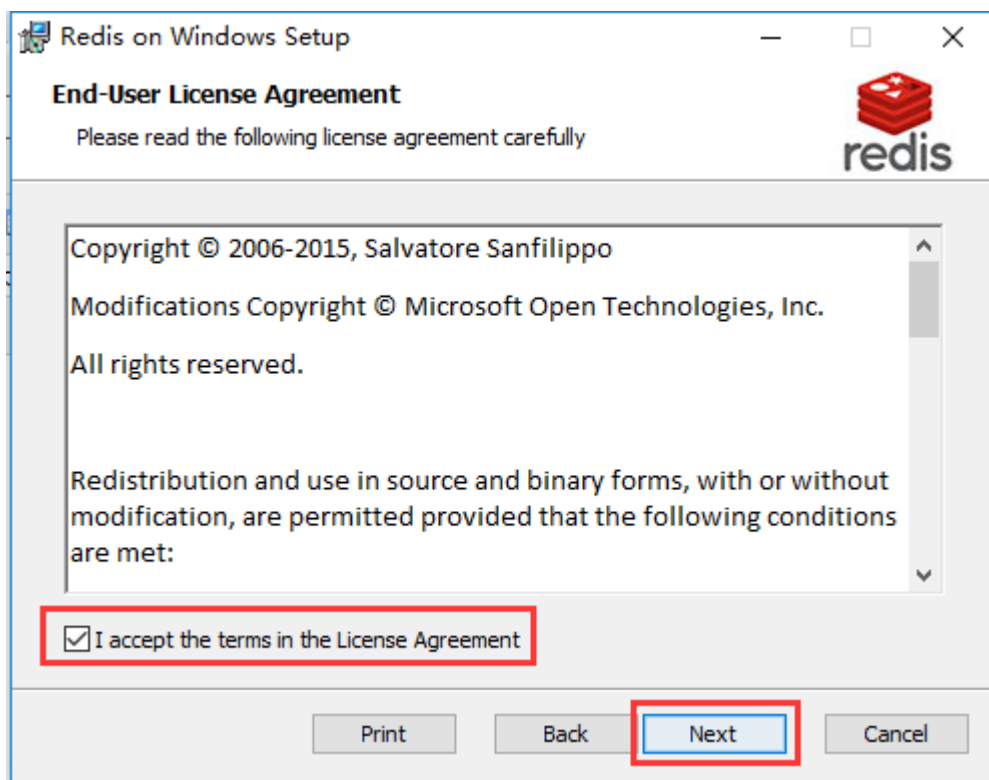
中文官网:<http://www.redis.cn>

### 5.1 redis下载和安装

下载地址: <https://github.com/MicrosoftArchive/redis/releases>







Redis on Windows Setup

**Port Number and Firewall Exception**

Select whether to add an exception to the Windows Firewall for Redis.

Port to run Redis on:

redis默认端口

☒ Add an exception to the Windows Firewall.  
让你的系统防火墙为redis放行

Back Next Cancel

Redis on Windows Setup

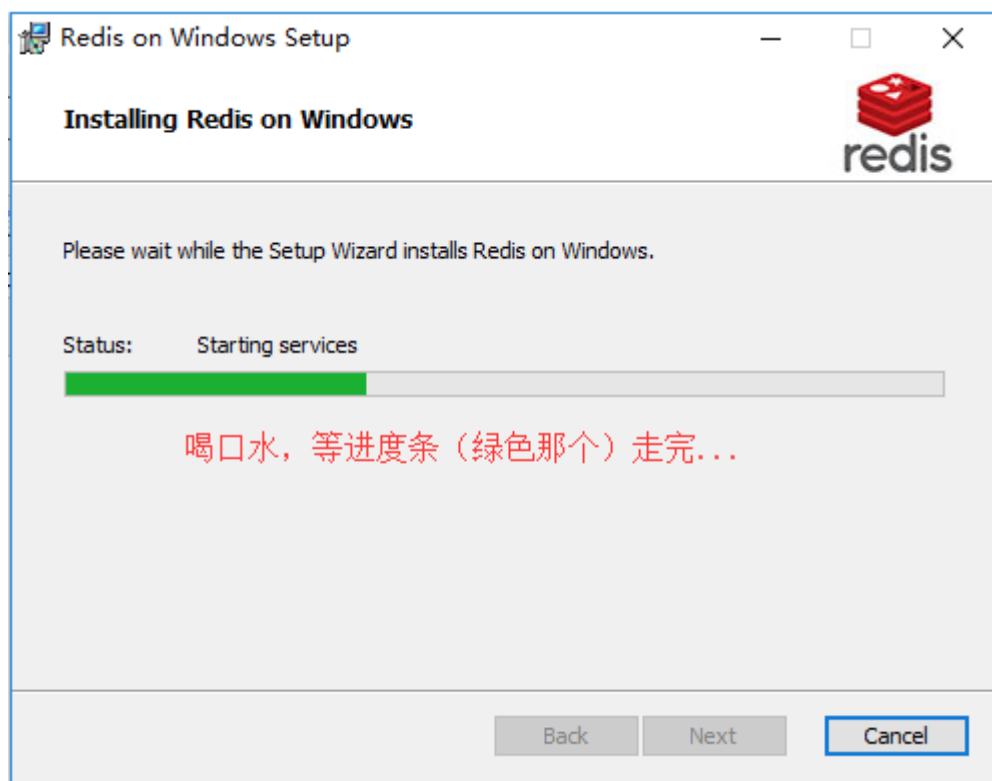
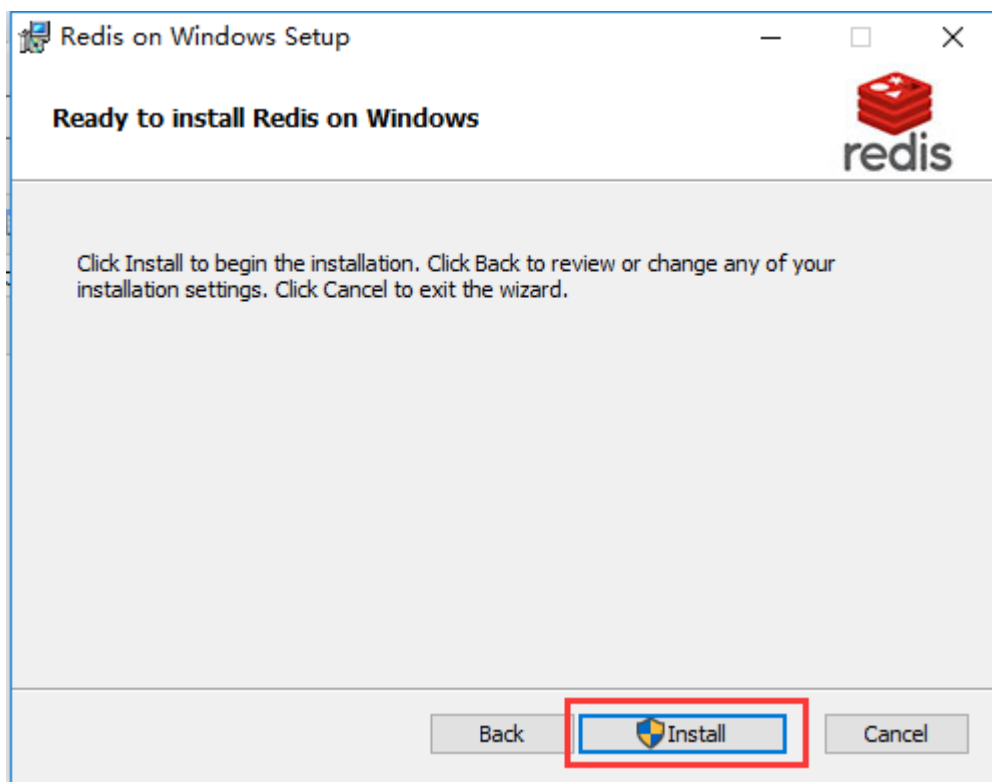
**Memory Limit**

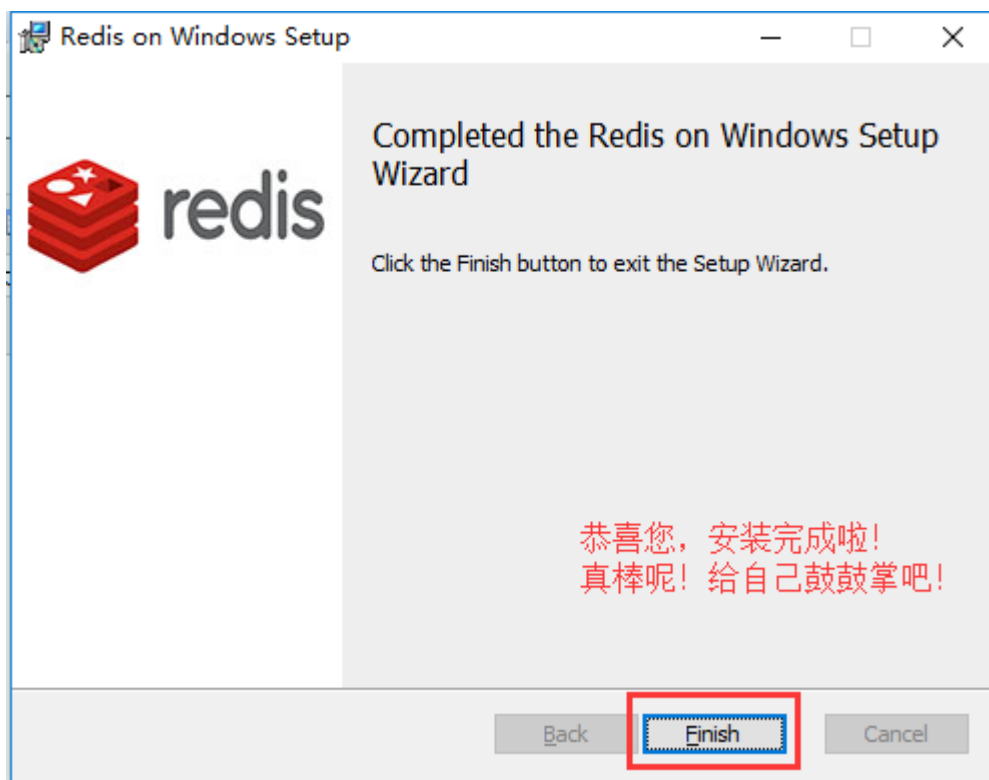
Select whether to use a memory limit or not.

☐ Set the Max Memory limit  
勾选这个，可以在下方修改redis占用的内存限制

Max Memory:  MB

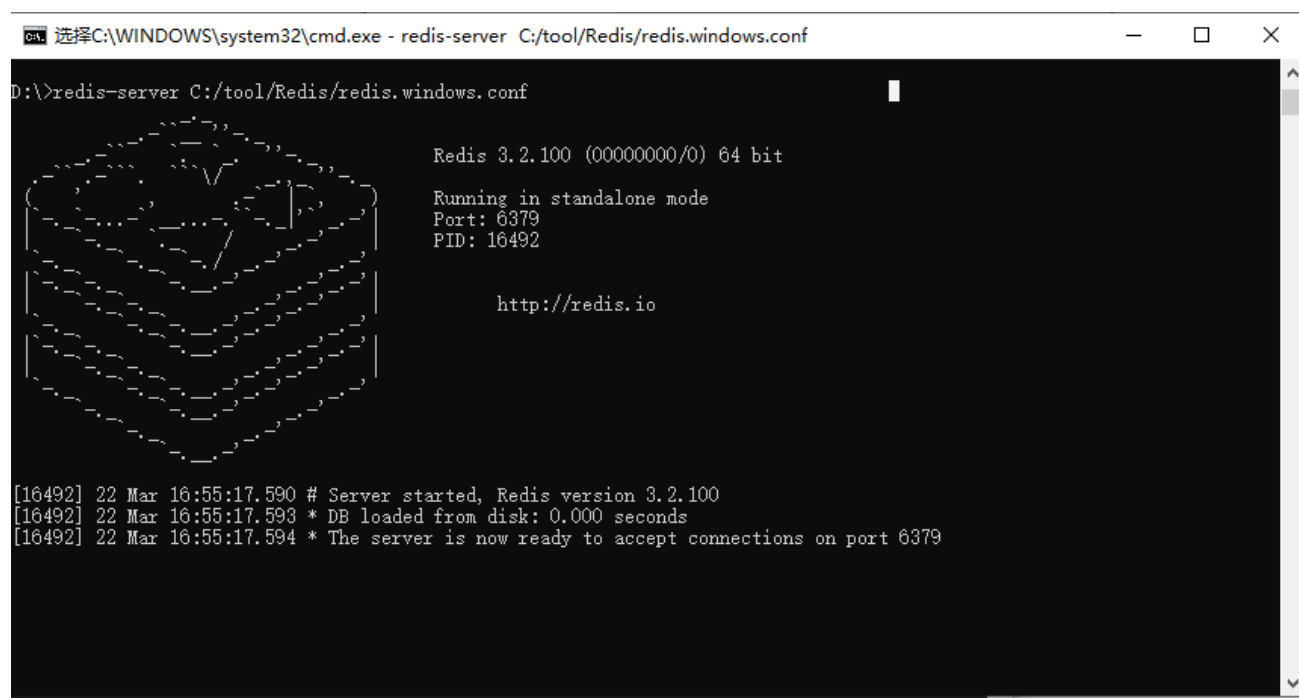
Back Next Cancel





使用以下命令启动redis服务端

```
redis-server C:/tool/redis/redis.windows.conf
```



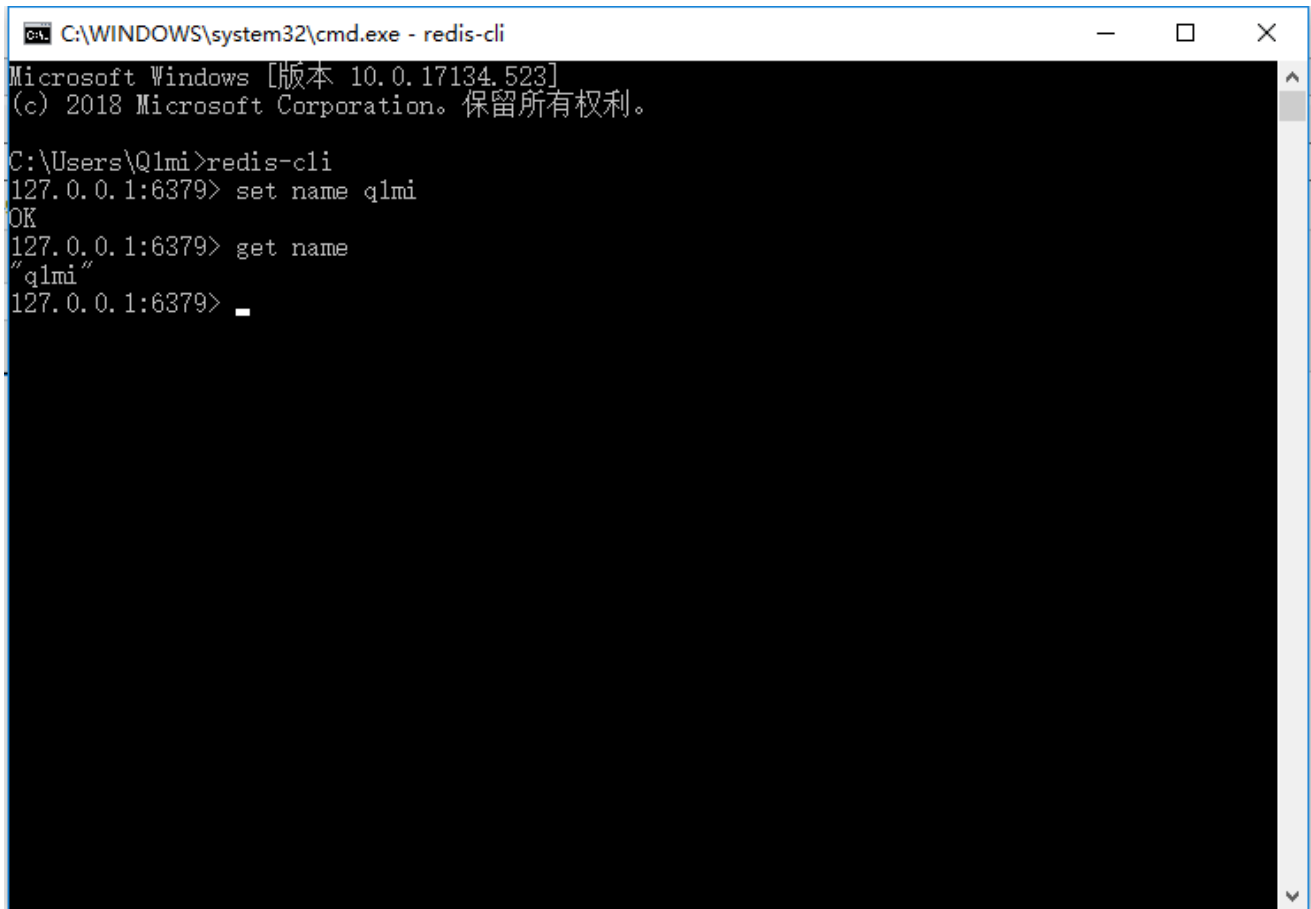
关闭上面这个cmd窗口就关闭redis服务器服务了。

### redis作为windows服务启动方式

```
redis-server --service-install redis.windows.conf
```

启动服务：redis-server --service-start 停止服务：redis-server --service-stop

启动内置客户端连接redis服务：



```
C:\WINDOWS\system32\cmd.exe - redis-cli
Microsoft Windows [版本 10.0.17134.523]
(c) 2018 Microsoft Corporation。保留所有权利。

C:\Users\Q1mi>redis-cli
127.0.0.1:6379> set name q1mi
OK
127.0.0.1:6379> get name
"q1mi"
127.0.0.1:6379> _
```

## 5.2 redis的使用

redis 安装成功以后,window下的配置文件保存在软件 安装目录下,如果是mac或者linux,则默认安装/etc/redis/redis.conf

### 5.2.1 redis的核心配置选项

- 绑定ip: 如果需要远程访问, 可将此行注释, 或绑定一个真实ip

```
bind 127.0.0.1
```
- 端口, 默认为6379

```
port 6379
```
- 是否以守护进程运行[这里的配置主要是linux和mac下面需要配置的]
  - 如果以守护进程运行, 则不会在命令行阻塞, 类似于服务
  - 如果以非守护进程运行, 则当前终端被阻塞
  - 设置为yes表示守护进程, 设置为no表示非守护进程

- 推荐设置为yes

```
daemonize yes
```

- 数据文件

```
dbfilename dump.rdb
```

- 数据文件存储路径

```
dir .
```

- 日志文件

```
logfile "C:/tool/redis/redis-server.log"
```

- 数据库，默认有16个

```
database 16
```

- 主从复制，类似于双机备份。

```
slaveof
```

## Redis

Redis 是一个高性能的key-value数据格式的内存缓存，NoSQL数据库。

NOSQL: not only sql, 泛指非关系型数据库。

关系型数据库: (mysql, oracle, sql server, sqlite)

1. 数据存放在表中，表之间有关系。
2. 通用的SQL操作语言。
3. 大部分支持事务。

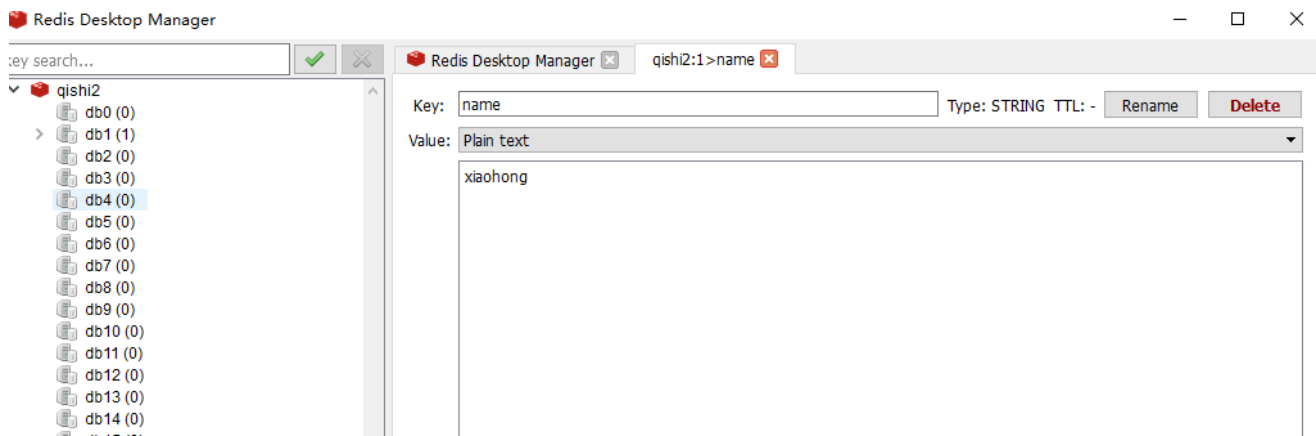
非关系型数据库[ redis, hadoop, mangoDB]:

1. 没有数据表的概念，不同的nosql数据库存放数据位置不同。
2. nosql数据库没有通用的操作语言。
3. 基本不支持事务。      redis支持简单事务

redis: 内存型(数据存放在内存中)的非关系型(nosql)key-value(键值存储)数据库，支持数据的持久化(注: 数据持久化时将数据存放到文件中，每次启动redis之后会先将文件中数据加载到内存)，经常用来做缓存(用来缓存一些经常用到的数据，提高读写速度)。

redis是一款基于CS架构的数据库，所以redis有客户端，也有服务端。

其中，客户端可以使用python等编程语言，也可以终端命令行工具



redis客户端连接服务器:

```
redis-cli -h `redis服务器ip` -p `redis服务器port`
```

## 5.3 redis数据类型

### 1. string类型:

字符串类型是 Redis 中最为基础的数据存储类型,它在 Redis 中是二进制安全的,也就是byte类型最大容量是512M。

### 2. hash类型:

hash用于存储对象,对象的结构为属性、值,值的类型为string。

```
key:{  
  域:值,  
  域:值,  
  域:值,  
  域:值,  
  ...  
}
```

### 3. list类型:

列表的元素类型为string。

key:[ 值1, 值2,值3..... ]

### 4. set类型:

无序集合,元素为string类型,元素唯一不重复,没有修改操作。

### 5. zset类型:

有序集合,元素为string类型,元素唯一不重复,没有修改操作。

## 5.4 string

如果设置的键不存在则为添加,如果设置的键已经存在则修改

- 设置键值

```
set key value
```

- 例1: 设置键为 `name` 值为 `xiaoming` 的数据

```
set name xiaoming
```

- 设置键值及过期时间，以秒为单位

```
setex key seconds value
```

- 例2：设置键为 `aa` 值为 `aa` 过期时间为3秒的数据

```
setex aa 3 aa
```

## 关于设置保存数据的有效期

```
# setex 添加保存数据到redis，同时设置有效期
```

格式：

```
setex key time value
```

```
# expire 给已有的数据重新设置有效期
```

格式：

```
expire key time
```

- 设置多个键值

```
mset key1 value1 key2 value2 ...
```

- 例3：设置键为 `a1` 值为 `python`、键为 `a2` 值为 `java`、键为 `a3` 值为 `c`

```
mset a1 python a2 java a3 c
```

- 追加值

```
append key value
```

- 例4：向键为 `a1` 中追加值 `haha`

```
append a1 haha
```

- 获取：根据键获取值，如果不存在此键则返回 `nil`

```
get key
```

- 例5：获取键 `name` 的值

```
get name
```

- 根据多个键获取多个值

```
mget key1 key2 ...
```

- 例6：获取键 `a1`、`a2`、`a3` 的值

```
mget a1 a2 a3
```



## 5.5 键操作

- 查找键，参数支持正则表达式

```
keys pattern
```

- 例1：查看所有键

```
keys *
```

- 例2：查看名称中包含 a 的键

```
keys a*
```

- 判断键是否存在，如果存在返回 1，不存在返回 0

```
exists key1
```

- 例3：判断键 a1 是否存在

```
exists a1
```

- 查看键对应的 value 的类型

```
type key
```

- 例4：查看键 a1 的值类型，为redis支持的五种类型中的一种

```
type a1
```

- 删除键及对应的值

```
del key1 key2 ...
```

- 例5：删除键 a2、a3

```
del a2 a3
```

- 设置过期时间，以秒为单位

- 如果没有指定过期时间则一直存在，直到使用 DEL 移除

```
expire key seconds
```

- 例6：设置键 a1 的过期时间为3秒

```
expire a1 3
```

- 查看有效时间，以秒为单位

```
ttl key
```

- 例7：查看键 bb 的有效时间

```
ttl bb
```

## 5.6 hash

- 设置单个属性

```
hset key field value
```

- 例1: 设置键 `user` 的属性 `name` 为 `xiaohong`

```
hset user name xiaohong
```

- 设置多个属性

```
hmset key field1 value1 field2 value2 ...
```

- 例2: 设置键 `u2` 的属性 `name` 为 `xiaohong`、属性 `age` 为 `11`

```
hmset u2 name xiaohong age 11
```

- 获取指定键所有的属性

```
hkeys key
```

- 例3: 获取键 `u2` 的所有属性

```
hkeys u2
```

- 获取一个属性的值

```
hget key field
```

- 例4: 获取键 `u2` 属性 `name` 的值

```
hget u2 name
```

- 获取多个属性的值

```
hmget key field1 field2 ...
```

- 例5: 获取键 `u2` 属性 `name`、`age` 的值

```
hmget u2 name age
```

- 获取所有属性的值

```
hvals key
```

- 例6: 获取键 `u2` 所有属性的值

```
hvals u2
```

- 删除属性，属性对应的值会被一起删除

```
hdel key field1 field2 ...
```

- 例7: 删除键 `u2` 的属性 `age`

```
hdel u2 age
```

## 5.7 list

列表的元素类型为string

## 按照插入顺序排序

- 在左侧插入数据

```
lpush key value1 value2 ...
```

- 例1: 从键为 `a1` 的列表左侧加入数据 `a`、`b`、`c`

```
lpush a1 a b c
```

- 在右侧插入数据

```
rpush key value1 value2 ...
```

- 例2: 从键为 `a1` 的列表右侧加入数据 `0`、`1`

```
rpush a1 0 1
```

- 在指定元素的前或后插入新元素

```
linsert key before或after 现有元素 新元素
```

- 例3: 在键为 `a1` 的列表中元素 `b` 前加入 `3`

```
linsert a1 before b 3
```

## 设置指定索引位置的元素值

- 索引从左侧开始，第一个元素为0
- 索引可以是负数，表示尾部开始计数，如 `-1` 表示最后一个元素

```
lset key index value
```

- 例5: 修改键为 `a1` 的列表中下标为 `1` 的元素值为 `z`

```
lset a1 1 z
```

- 删除指定元素

- 将列表中前 `count` 次出现的值为 `value` 的元素移除
- `count > 0`: 从头往尾移除
- `count < 0`: 从尾往头移除
- `count = 0`: 移除所有

```
lrem key count value
```

- 例6.1: 向列表 `a2` 中加入元素 `a`、`b`、`a`、`b`、`a`、`b`

```
lpush a2 a b a b a b
```

- 例6.2: 从 `a2` 列表右侧开始删除2个 `b`

```
lrem a2 -2 b
```

- 例6.3: 查看列表 `a2` 的所有元素

```
lrange a2 0 -1
```

## 5.8 set

- 添加元素

```
sadd key member1 member2 ...
```

- 例1: 向键 `a3` 的集合中添加元素 `zhangsan`、`lisi`、`wangwu`

```
sadd a3 zhangsan sili wangwu
```

- 返回所有的元素

```
smembers key
```

- 例2: 获取键 `a3` 的集合中所有元素

```
smembers a3
```

- 删除指定元素

```
srem key
```

- 例3: 删除键 `a3` 的集合中元素 `wangwu`

```
srem a3 wangwu
```

## 5.9 关于redis的几个站点地址

中文官网: <http://www.redis.cn/>

英文官网: <https://redis.io>

参考命令: <http://doc.redisfans.com/>

### 针对redis中的内容扩展

`flushall` 清空当前数据库中的所有数据

针对各种数据类型它们的特性, 使用场景如下:

字符串string: 用于保存一些项目中的普通数据, 只要键值对的都可以保存, 例如, 保存 session, 定时记录状态

哈希hash: 用于保存项目中的字典数据, 但是不能保存多维的字典, 例如, 商城的购物车

列表list: 用于保存项目中的列表数据, 但是也不能保存多维的列表, 例如, 队列, 秒杀, 医院的挂号

无序集合set: 用于保存项目中的不能重复的数据, 可以用于过滤, 例如, 投票海选的时候, 过滤候选人

有序集合zset: 用于保存项目中一些不能重复, 但是需要进行排序的数据, 分数排行榜。

