# **项目基础及工具**

| **Tedu Python 教学部** |
| --- |
| Author：吕泽 |
| Days：3天 |

* [GIT简介](#git%E7%AE%80%E4%BB%8B)
* [GIT使用](#git%E4%BD%BF%E7%94%A8)
  + [基本概念](#%E5%9F%BA%E6%9C%AC%E6%A6%82%E5%BF%B5)
  + [初始配置](#%E5%88%9D%E5%A7%8B%E9%85%8D%E7%BD%AE)
  + [基本命令](#%E5%9F%BA%E6%9C%AC%E5%91%BD%E4%BB%A4)
    - [@扩展延伸](#%E6%89%A9%E5%B1%95%E5%BB%B6%E4%BC%B8)
  + [版本控制](#%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6)
  + [保存工作区](#%E4%BF%9D%E5%AD%98%E5%B7%A5%E4%BD%9C%E5%8C%BA)
  + [分支管理](#%E5%88%86%E6%94%AF%E7%AE%A1%E7%90%86)
  + [远程仓库](#%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93)
    - [远程仓库操作命令](#%E8%BF%9C%E7%A8%8B%E4%BB%93%E5%BA%93%E6%93%8D%E4%BD%9C%E5%91%BD%E4%BB%A4)
* [GitHub使用](#github%E4%BD%BF%E7%94%A8)
  + [介绍](#%E4%BB%8B%E7%BB%8D)
  + [获取项目](#%E8%8E%B7%E5%8F%96%E9%A1%B9%E7%9B%AE)
  + [创建git仓库](#%E5%88%9B%E5%BB%BAgit%E4%BB%93%E5%BA%93)
  + [添加ssh秘钥](#%E6%B7%BB%E5%8A%A0ssh%E7%A7%98%E9%92%A5)
* [PIP的使用](#pip%E7%9A%84%E4%BD%BF%E7%94%A8)
* [软件项目开发流程](#%E8%BD%AF%E4%BB%B6%E9%A1%B9%E7%9B%AE%E5%BC%80%E5%8F%91%E6%B5%81%E7%A8%8B)
  + [项目注意事项](#%E9%A1%B9%E7%9B%AE%E6%B3%A8%E6%84%8F%E4%BA%8B%E9%A1%B9)
  + [项目工具的使用](#%E9%A1%B9%E7%9B%AE%E5%B7%A5%E5%85%B7%E7%9A%84%E4%BD%BF%E7%94%A8)
* [电子词典](#%E7%94%B5%E5%AD%90%E8%AF%8D%E5%85%B8)
  + [功能说明](#%E5%8A%9F%E8%83%BD%E8%AF%B4%E6%98%8E)
* [HTTPServer](#httpserver)
  + [功能 ：](#%E5%8A%9F%E8%83%BD)

## **GIT简介**

1. 什么是GIT

git是一个开源的分布式版本控制系统，用于高效的管理各种大小项目和文件。

1. 代码管理工具的用途

* 防止代码丢失，做备份
* 项目的版本管理和控制，可以通过设置节点进行跳转
* 建立各自的开发环境分支，互不影响，方便合并
* 在多终端开发时，方便代码的相互传输

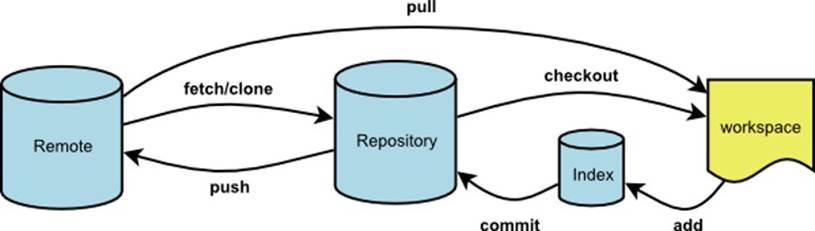
1. git的特点

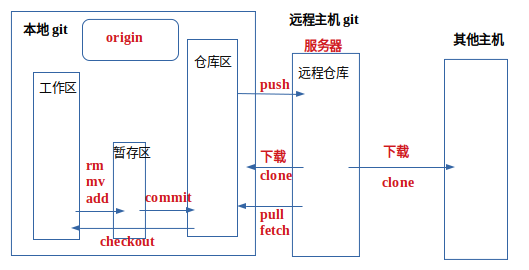
* git是开源的，多在\*nix下使用，可以管理各种文件
* git是分布式的项目管理工具(svn是集中式的)
* git数据管理更多样化，分享速度快，数据安全
* git 拥有更好的分支支持，方便多人协调

1. git安装

sudo apt-get install git

## **GIT使用**





### **基本概念**

* 工作区：项目所在操作目录，实际操作项目的区域
* 暂存区: 用于记录工作区的工作（修改）内容
* 仓库区: 用于备份工作区的内容
* 远程仓库: 远程主机上的GIT仓库

注意： 在本地仓库中，git总是希望工作区的内容与仓库区保持一致，而且只有仓库区的内容才能和其他远程仓库交互。

### **初始配置**

配置命令: git config

* 配置所有用户： git config --system [选项]

配置文件位置: /etc/gitconfig

* 配置当前用户： git config --global [选项]

配置文件位置: ~/.gitconfig

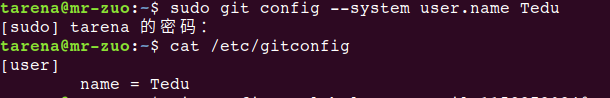
* 配置当前项目： git config [选项]

配置文件位置: project/.git/config

1. **配置用户名**

e.g. 将用户名设置为Tedu

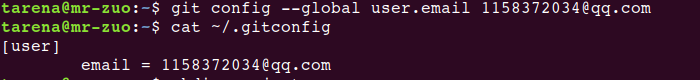
sudo git config --system user.name Tedu



1. **配置用户邮箱**

e.g. 将邮箱设置为lvze@tedu.cn

git config --global user.email [lvze@tedu.cn](mailto:lvze@tedu.cn)



1. **配置编译器**

e.g. 配置编译器为pycharm

git config core.editor pycharm

1. **查看配置信息**

git config --list

### **基本命令**

1. **初始化仓库**

git init  
意义：将某个项目目录变为git操作目录，生成git本地仓库。即该项目目录可以使用git管理

1. **查看本地仓库状态**

git status  
说明: 初始化仓库后默认工作在master分支，当工作区与仓库区不一致时会有提示。

1. **将工作内容记录到暂存区**

git add [files..]

e.g. 将 a ，b 记录到暂存区

git add a b

e.g. 将所有文件（不包含隐藏文件）记录到暂存区

git add \*

1. **取消文件暂存记录**

git rm --cached [file]

1. **将文件同步到本地仓库**

git commit [file] -m [message]  
说明: -m表示添加一些同步信息，表达同步内容

e.g. 将暂存区所有记录同步到仓库区

git commit -m 'add files'

1. **查看commit 日志记录**

git log  
git log --pretty=oneline

1. **比较工作区文件和仓库文件差异**

git diff [file]

1. **放弃工作区文件修改**

git checkout -- [file]

1. **从仓库区恢复文件**

git checkout [file]

**10 移动或者删除文件**

git mv [file] [path]  
git rm [files]  
注意: 这两个操作会修改工作区内容，同时将操作记录提交到暂存区。

#### **@扩展延伸**

在Git项目中可以通过在项目的某个文件夹下定义.gitignore文件的方式，规定相应的忽略规则，用来管理当前文件夹下的文件的Git提交行为。.gitignore 文件是可以提交到公有仓库中，这就为该项目下的所有开发者都共享一套定义好的忽略规则。在.gitingore 文件中，遵循相应的语法，在每一行指定一个忽略规则。

.gitignore忽略规则简单说明

`#` 表示此为注释,将被Git忽略

\*.a 表示忽略所有 .a 结尾的文件

!lib.a 表示但lib.a除外

/TODO 表示仅仅忽略项目根目录下的 TODO 文件，不包括 subdir/TODO

build/ 表示忽略 build/目录下的所有文件，过滤整个build文件夹；

doc/\*.txt 表示会忽略doc/notes.txt但不包括 doc/server/arch.txt

bin/: 表示忽略当前路径下的bin文件夹，该文件夹下的所有内容都会被忽略，不忽略 bin 文件

/bin: 表示忽略根目录下的bin文件

/\*.c: 表示忽略cat.c，不忽略 build/cat.c

debug/\*.obj: 表示忽略debug/io.obj，不忽略 debug/common/io.obj和tools/debug/io.obj

\*\*/foo: 表示忽略/foo,a/foo,a/b/foo等

a/\*\*/b: 表示忽略a/b, a/x/b,a/x/y/b等

!/bin/run.sh 表示不忽略bin目录下的run.sh文件

\*.log: 表示忽略所有 .log 文件

config.php: 表示忽略当前路径的 config.php 文件

/mtk/ 表示过滤整个文件夹

\*.zip 表示过滤所有.zip文件

/mtk/do.c 表示过滤某个具体文件

### **版本控制**

1. **退回到上一个commit节点**

git reset --hard HEAD^  
注意 ： **一个^**表示回退1个版本，依次类推。当版本回退之后工作区会自动和当前commit版本保持一致

1. **退回到指定的commit\_id节点**

git reset --hard [commit\_id]

1. **查看所有操作记录**

**git reflog**  
注意:最上面的为最新记录，可以利用commit\_id(前七位数字)去往任何操作位置

1. **创建标签**

标签: 在项目的重要commit位置添加快照，保存当时的工作状态，一般用于版本的迭代。

**git tag [tag\_name] [commit\_id] -m [message]**  
说明: commit\_id可以不写则默认标签表示最新的commit\_id位置，message也可以不写，但是最好添加。

e.g. 在最新的commit处添加标签v1.0

git tag v1.0 -m '版本1'

1. **查看标签**

git tag 查看标签列表  
git show [tag\_name] 查看标签详细信息

1. **去往某个标签节点**

git reset --hard [tag]

1. **删除标签**

git tag -d [tag]

### **保存工作区**

1. **保存工作区内容**

git stash save [message]  
说明: 将工作区未提交的修改封存，让工作区回到修改前的状态

1. **查看工作区列表**

git stash list  
说明:最新保存的工作区在最上面

1. **应用某个工作区**

git stash apply [stash@{n}]

1. **删除工作区**

git stash drop [stash@{n}] 删除某一个工作区  
git stash clear 删除所有保存的工作区

### **分支管理**

定义: 分支即每个人在原有代码（分支）的基础上建立自己的工作环境，单独开发，互不干扰。完成开发工作后再进行分支统一合并。

1. **查看分支情况**

git branch  
说明: 前面带 **\*** 的分支表示当前工作分支

1. **创建分支**

git branch [branch\_name]  
说明: 基于a分支创建b分支，此时b分支会拥有a分支全部内容。在创建b分支时最好保持a分支"干净"状态。

1. **切换工作分支**

git checkout [branch]  
说明: 2,3可以同时操作，即创建并切换分支

git checkout -b [branch\_name]

1. **合并分支**

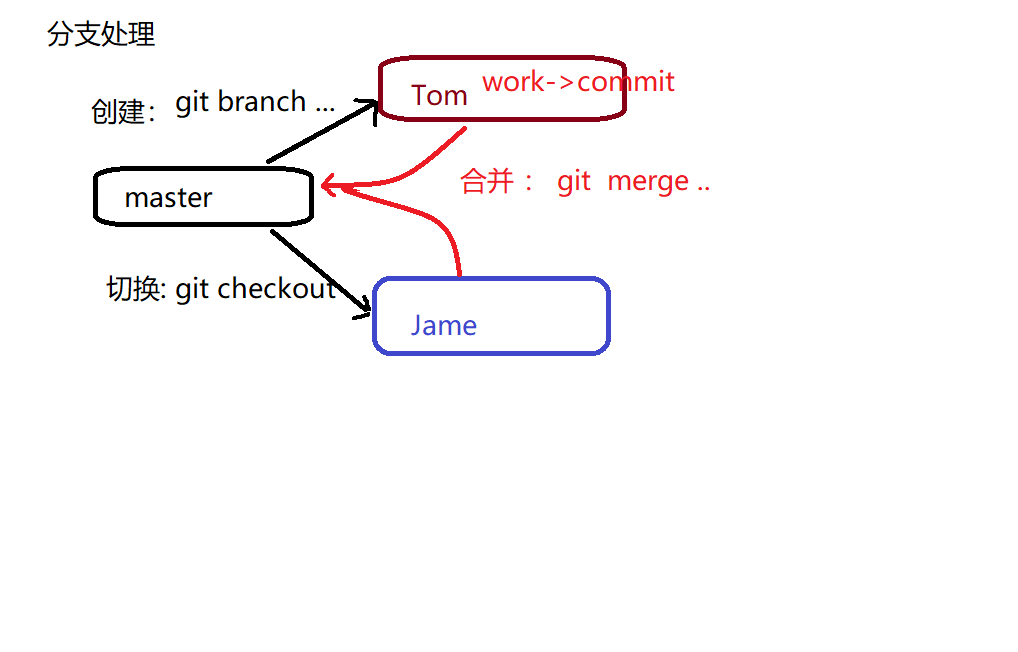
git merge [branch]

冲突问题是合并分支过程中最为棘手的问题

当分支合并时，原分支和以前发生了变化就会产生冲突  
当合并分支时添加新的模块（文件），这种冲突可以自动解决，只需自己决定commit操作即可。  
当合并分支时两个分支修改了同一个文件，则需要手动解决冲突。

1. **删除分支**

git branch -d [branch] 删除分支  
git branch -D [branch] 删除没有被合并的分支



### **远程仓库**

1. 什么是远程仓库

远程主机上的git仓库。实际上git是分布式结构，每台主机的git仓库结构类似，只是把别人主机上的git仓库称为远程仓库。

1. 共享仓库

在git仓库中bare属性为**True**的共享仓库可以很好的和远程仓库进行交互

创建步骤：

* 选择共享仓库目录，将该目录属主设置为当前用户

mkdir gitrepo

chown tarena:tarena gitrepo

* 将该目录初始化为git共享目录，下例中tedu为自己取的项目名称，.git为通用结尾后缀

cd gitrepo

git init --bare tedu.git

* 将git配置目录与项目目录设置为相同的属主

chown -R tarena:tarena tedu.git

#### **远程仓库操作命令**

所有操作在本地git仓库下进行

1. **添加远程仓库**

git remote add origin tarena@127.0.0.1:/home/tarena/gitrepo/tedu.git

1. **删除远程主机**

git remote rm [origin]

1. **查看连接的主机**

git remote  
注意: 一个git项目连接的远程主机名不会重复

1. **将本地分支推送给远程仓库（上传）**

将master分支推送给origin主机远程仓库，第一次推送分支使用-u表示与远程对应分支建立自动关联

**git push -u origin master**

1. **删除远程分支**

git branch -a 查看所有分支  
git push origin [:branch] 删除远程分支

1. **其他推送方法**

git push --force origin 用于本地版本比远程版本旧时强行推送本地版本

git push origin [tag] 推送本地标签到远程

git push origin --tags 推送本地所有标签到远程

git push origin --delete tag [tagname] 删除远程仓库标签

1. **从远程获取项目**

**git clone tarena@127.0.0.1:/home/tarena/gitrepo/tedu.git**

注意： 获取到本地的项目会自动和远程仓库建立连接。且获取的项目本身也是个git项目。

1. **从远程获取代码（下载）**

**git pull**

将远程分支master拉取到本地，作为tmp分支  
**git fetch origin master:tmp**

区别

pull将远程内容直接拉取到本地，并和对应分支内容进行合并  
fetch将远程分支内容拉取到本地，但是不会和本地对应分支合并，可以自己判断后再使用merge合并。

## **GitHub使用**

### **介绍**

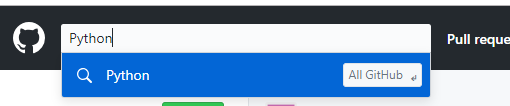
github是一个开源的项目社区网站，拥有全球最多的开源项目。开发者可以注册网站在github建立自己的项目仓库。

网址： [github.com](http://github.com/)

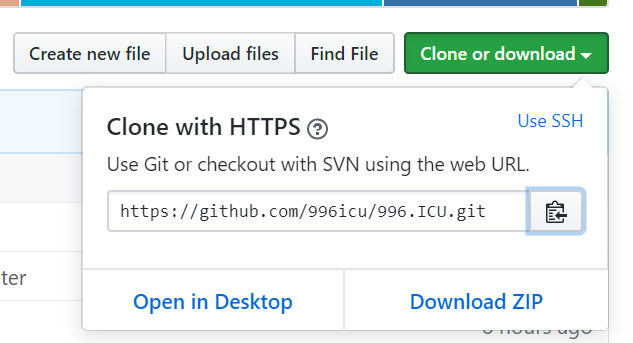
代码管理工具：git

### **获取项目**

* 在左上角搜索栏搜索想要的获取的项目



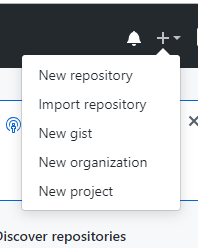
* 选择项目后复制项目git地址



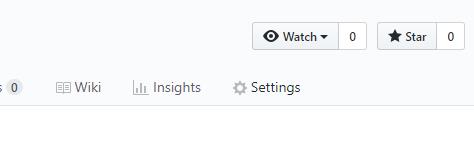
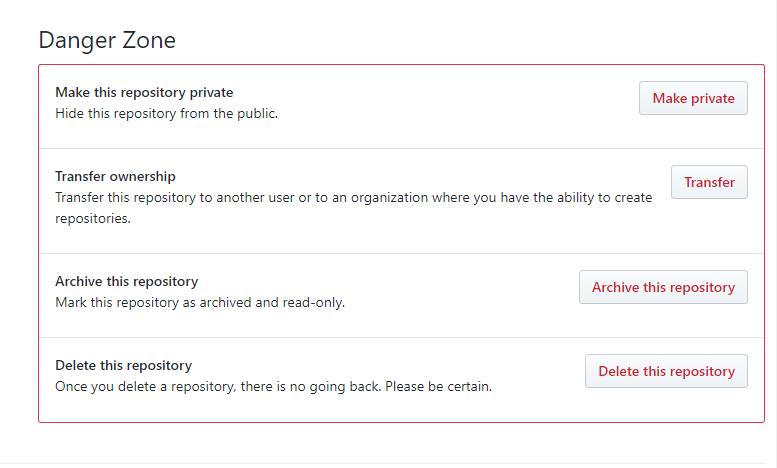
* 在本地使用git clone方法即可获取

### **创建git仓库**

* 点击右上角加号下拉菜单，选择新的仓库



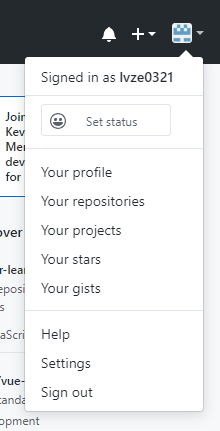
* 填写相应的项目信息即可
* github仓库相对本地主机就是一个远程仓库 通过remote连接，如果需要输入密码输 入github密码即可。连接后即可使用远程仓库操作命令操作。readme文件会被自动作为项目介绍
* 如果是在即的仓库在仓库界面选择settings，在最后可以选择删除仓库

### **添加ssh秘钥**

如果使用http协议访问github则每次都需要输入密码，如果是自己的私人计算机可以使用ssh协议访问，添加秘钥后则不再需要每次输入密码。

* 登录github账号
* 右上角头像下拉菜单--》settings --》左侧 SSH and GPG keys --》new ssh key --》填写title，将私人计算机公钥内容加入key文本框 点击add...



## **PIP的使用**

1. 作用 ： 管理python的**标准第三方库**中第三方软件包
2. 安装

sudo apt-get install python3-pip

1. 常用命令：

安装软件： pip3 install [package]

e.g. sudo pip3 install ssh

查看当前python软件包 ： pip3 list

搜索某个名字的python包：pip3 search [name]

查看软件包信息：pip3 show [package]

升级软件包： pip3 install --upgrade [package]

卸载软件包： sudo pip3 uninstall [package]

e.g. sudo pip3 uninstall ssh

导出软件包环境：pip3 freeze > requirements.txt

根据文档自动安装:pip3 install -r requirements.txt

## **软件项目开发流程**

需求分析 ----》 概要设计 ---》 项目计划 ----》详细设计---》编码测试 -----》项目测试 ----》调试修改 ---》项目发布----》后期维护

需求分析 ： 确定用户的真实需求

1. 确定用户的真实需求，项目的基本功能
2. 确定项目的整体难度和可行性分析
3. 需求分析文档，用户确认

概要设计：对项目进行初步分析和整体设计

1. 确定功能模块
2. 进行可行性分析 搭建整体架构图
3. 确定技术思路和使用框架
4. 形成概要文档指导开发流程

项目计划 ： 确定项目开发的时间轴和流程

1. 确定开发工作的先后顺序
2. 确定时间轴 ，事件里程碑
3. 人员分工
4. 形成甘特图和思维导图等辅助内容

详细设计 ： 项目的具体实现手册

1.形成详细设计文档 ： 思路，逻辑流程，功能说明，技术点说明，数据结构说明，代码说明

编码测试 ： 按照预定计划实现代码编写，并且做基本检测

1. 代码编写
2. 写测试程序
3. 技术攻关

项目测试 ： 对项目按照公进行测试

1. 跨平台测试 ，使用测试
2. 根据测试报告进行代码修改
3. 完成测试报告

项目发布

1.项目交付用户进行发布  
2.编写项目说明文档

后期维护

1.维护项目正常运转  
2.进行项目的迭代升级

### **项目注意事项**

* 按时完成项目工作和项目时间不足之间的冲突
* 项目实施人员之间的冲突

### **项目工具的使用**

编写文档： word ppt Excel markdown LaTex  
项目流程图 ： Mindmanager visio  
项目管理 ： project  
代码管理 ： Svn git

## **电子词典项目**

参考代码：dict

### **功能说明**

用户可以登录和注册  
\* 登录凭借用户名和密码登录  
\* 注册要求用户必须填写用户名，密码，其他内容自定  
\* 用户名要求不能重复  
\* 要求用户信息能够长期保存

可以通过基本的图形界面print以提示客户端输入。  
\* 程序分为服务端和客户端两部分  
\* 客户端通过print打印简单界面输入命令发起请求  
\* 服务端主要负责逻辑数据处理  
\* 启动服务端后应该能满足多个客户端同时操作

客户端启动后即进入一级界面，包含如下功能：登录 注册 退出

\* 退出后即退出该软件

\* 登录成功即进入二级界面，失败回到一级界面

\* 注册成功可以回到一级界面继续登录，也可以直接用注册用户进入二级界面

用户登录后进入二级界面，功能如下：查单词 历史记录 注销

\* 选择注销则回到一级界面

\* 查单词：循环输入单词，得到单词解释，输入特殊符号退出单词查询状态

\* 历史记录：查询当前用户的查词记录，要求记录包含name word time。可以查看所有记录或者前10条均可。

单词本说明

每个单词一定占一行  
单词按照从小到大顺序排列  
单词和解释之间一定有空格

查词说明

直接使用单词本查询（文本操作）  
先将单词存入数据库，然后通过数据库查询。（数据库操作）

### 分析：

1. 确定技术

通信 tcp通信

并发 多进程并发

数据库 mysql

2. 确定数据库 ： 建立几个表，每个表作用和存储内容

create database dict charset=utf8;

\* 建表

用户表 ： id name passwd

create table user (id int primary key auto\_increment,name varchar(32) not null,passwd varchar(128) not null);

历史记录：id name word time

create table hist (id int primary key auto\_increment,name varchar(32) not null,word varchar(32) not null,time varchar(64) not null);

单词表： id word mean

create table words (id int primary key auto\_increment,word varchar(32),mean text);

\* 编写程序将单词本存入到数据库

3. 结构设计

客户端

服务端 （处理数据）

4. 功能分析

客户端和服务端分别需要实现哪些功能

网络模型

注册

客户端 \* 输入注册信息

\* 将信息发送给服务端

\* 等待反馈

服务端 \* 接收注册信息

\* 验证用户是否存在

\* 插入数据库

\* 将信息反馈个客户端

登录

客户端 \* 输入登录信息

\* 发送请求

\* 得到回复

服务端 \* 接收请求

\* 判断是否允许登录

\* 反馈结果

查单词

历史记录

协议指定 ： 注册 R name passwd

登录 L name passwd

cookie01 ：

import getpass

getpass.getpass()

功能: 隐藏输入内容(将要输入的内容隐藏)

返回值： 输入的内容字符串

cookie02 ：

import hashlib

　功能：对密码进行加密

生成加密对象 参数为 "盐"

hash = hashlib.md5(("Levi"+"the-sat").encode())

对密码进行算法加密

hash.update(passwd.encode())

获取加密后的密码字串

hash.hexdigest()

### 代码实现：

#### １．将字典插入数据库模块

*"""*

*将dict.txt的单词插入列表words当中*

*"""*

**import** pymysql

**import** re

f = open(**'dict.txt'**)

*# 连接数据库*

db = pymysql.connect(host=**'localhost'**,

port=3306,

user=**'root'**,

passwd=**'123456'**,

database=**'dict\_sql'**,

charset=**'utf8'**)

*# 获取游标*

cur = db.cursor()

sql = **'insert into words(word,mean) values(%s,%s)'**

**for** line **in** f:

tup = re.findall(**r'(\w+)\s+(.\*)'**, line)[0]

**try**:

*# 数据操作*

cur.execute(sql, tup)

*# 将修改内容提交到数据库*

db.commit()

**except** Exception **as** e:

print(e)

db.rollback()

*# 关闭游标和数据库连接*

cur.close()

db.cursor()

#### ２．数据库处理模块

*"""*

*dict项目用于处理数据*

*"""*

**import** pymysql

**import** hashlib

**import** time

*# 　编写功能类　提供给服务端使用*

**class** Database:

**def** \_\_init\_\_(self, host=**'localhost'**,

port=3306,

user=**'root'**,

passwd=**'123456'**,

database=**'dict\_sql'**, *# 数据库*

charset=**'utf8'**):

self.host = host

self.port = port

self.user = user

self.passwd = passwd

self.database = database

self.charset = charset

self.connect\_db() *# 链接数据库*

**def** connect\_db(self):

self.db = pymysql.connect(host=self.host,

port=self.port,

user=self.user,

passwd=self.passwd,

database=self.database,

charset=self.charset)

*# 创建游标*

**def** create\_cursor(self):

self.cur = self.db.cursor()

*# 　关闭数据库*

**def** close(self):

self.cur.close()

self.db.close()

*# 　处理注册*

**def** register(self, name, passwd):

sql = **"select \* from user where name='%s'"** % name

self.cur.execute(sql)

r = self.cur.fetchone() *# 如果查询到结果*

**if** r:

**return False**

*# 加密处理*

hash = hashlib.md5((name + **"the-salt"**).encode())

hash.update(passwd.encode())

sql = **"insert into user (name,passwd) values (%s,%s)"**

**try**:

self.cur.execute(sql, [name, hash.hexdigest()])

self.db.commit()

**return True**

**except** Exception:

self.db.rollback()

**return False**

*# 处理登录*

**def** login(self, name, passwd):

sql = **"select \* from user where \**

**name = %s and passwd = %s"**

*# 　name + the-salt -->盐*

hash = hashlib.md5((name + **"the-salt"**).encode())

hash.update(passwd.encode())

self.cur.execute(sql, [name, hash.hexdigest()])

r = self.cur.fetchone()

**if** r:

**return True**

**else**:

**return False**

*# 插入历史记录*

**def** insert\_history(self, name, word):

tm = time.ctime()

sql = **"insert into hist (name,word,time) \**

**values (%s,%s,%s)"**

**try**:

self.cur.execute(sql, [name, word, tm])

self.db.commit()

**except** Exception:

self.db.rollback()

*# 单词查询*

**def** query(self, word):

sql = **"select mean from words \**

**where word = '%s'"** % word

self.cur.execute(sql)

r = self.cur.fetchone()

**if** r:

**return** r[0]

*#　历史记录*

**def** history(self,name):

sql = **"select name,word,time from hist \**

**where name='%s' order by id desc limit 10"**%name

self.cur.execute(sql)

**return** self.cur.fetchall()

#### ３．服务端模块

*"""*

*dict 服务端部分*

*处理请求逻辑*

*dict\_server v 1.0*

*"""*

**from** socket **import** \*

**from** multiprocessing **import** Process

**import** signal

**import** sys

**from** time **import** sleep

**from** operation\_db **import** \*

*# 　全局变量*

HOST = **'0.0.0.0'**

PORT = 8000

ADDR = (HOST, PORT)

*# 　处理注册*

**def** do\_register(c, db, data):

tmp = data.split(**' '**)

name = tmp[1]

passwd = tmp[2]

**if** db.register(name, passwd):

c.send(**b'OK'**)

**else**:

c.send(**b"FAIL"**)

*# 　处理登录L name passwd*

**def** do\_login(c, db, data):

tmp = data.split(**' '**)

name = tmp[1]

passwd = tmp[2]

**if** db.login(name, passwd):

c.send(**b'OK'**)

**else**:

c.send(**b"FAIL"**)

*# 　处理查询Q name word*

**def** do\_query(c, db, data):

tmp = data.split(**' '**)

name = tmp[1]

word = tmp[2]

*# 插入历史记录*

db.insert\_history(name, word)

*# 　查单词 没查到返回Ｎｏｎｅ*

mean = db.query(word)

**if not** mean:

c.send(**"没有找到该单词"**.encode())

**else**:

msg = **"%s : %s"** % (word, mean)

c.send(msg.encode())

**def** do\_hist(c, db, data):

name = data.split(**' '**)[1]

r = db.history(name)

**if not** r:

c.send(**b"FAIL"**)

**return**

c.send(**b'OK'**)

**for** i **in** r:

*# i ==> (name,word,time)*

msg = **"%s %s %s"** % i

sleep(0.1) *# 防止沾包*

c.send(msg.encode())

sleep(0.1)

c.send(**b'##'**)

*# 　处理客户端请求*

**def** do\_request(c, db):

db.create\_cursor() *# 生成游标* *db.cur*

**while True**:

data = c.recv(1024).decode()

print(c.getpeername(), **':'**, data)

**if not** data **or** data[0] == **'E'**:

c.close()

sys.exit(**"客户端退出"**)

**elif** data[0] == **'R'**:

do\_register(c, db, data)

**elif** data[0] == **'L'**:

do\_login(c, db, data)

**elif** data[0] == **'Q'**:

do\_query(c, db, data)

**elif** data[0] == **'H'**:

do\_hist(c, db, data)

*# 　网络链接*

**def** main():

*# 　创建数据库链接对象*

db = Database()

*# 　创建ｔｃｐ套接字*

s = socket()

s.setsockopt(SOL\_SOCKET, SO\_REUSEADDR, 1)

s.bind(ADDR)

s.listen(5)

*# 处理僵尸进程*

signal.signal(signal.SIGCHLD, signal.SIG\_IGN)

*# 　等待客户端链接*

print(**"Listen the port 8000"**)

**while True**:

**try**:

c, addr = s.accept()

print(**"Connect from"**, addr)

**except** KeyboardInterrupt:

s.close()

db.close()

sys.exit(**"服务器退出"**)

**except** Exception **as** e:

print(e)

**continue**

*# 创建子进程*

p = Process(target=do\_request, args=(c, db))

p.daemon = **True**

p.start()

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

main()

#### ４．客户端模块

*"""*

*dict 客户端*

*发起请求，展示结果*

*"""*

**from** socket **import** \*

**from** getpass **import** getpass

ADDR = (**'127.0.0.1'**, 8000)

*# 　所有函数都用ｓ*

s = socket()

s.connect(ADDR)

*# 　查单词*

**def** do\_query(name):

**while True**:

word = input(**"单词:"**)

**if** word == **'##'**: *# 结束单词查询*

**break**

msg = **"Q %s %s"** % (name, word)

s.send(msg.encode())

*# 　等待回复*

data = s.recv(2048).decode()

print(data)

**def** do\_hist(name):

msg = **"H %s"** % name

s.send(msg.encode())

data = s.recv(128).decode()

**if** data == **'OK'**:

**while True**:

data = s.recv(1024).decode()

**if** data == **'##'**:

**break**

print(data)

**else**:

print(**"还没有历史记录"**)

*# 二级界面*

**def** login(name):

**while True**:

print(**"""**

**================Query=================**

**1. 查单词** **2. 历史记录　　　3. 注销**

**======================================**

**"""**)

cmd = input(**"输入选项:"**)

**if** cmd == **'1'**:

do\_query(name)

**elif** cmd == **'2'**:

do\_hist(name)

**elif** cmd == **'3'**:

**return**

**else**:

print(**"请输入正确命令!"**)

*# 注册*

**def** do\_register():

**while True**:

name = input(**"User:"**)

passwd = getpass()

passwd1 = getpass(**"Again:"**)

**if** (**' ' in** name) **or** (**' ' in** passwd):

print(**"用户名或密码不能有空格"**)

**continue**

**if** passwd != passwd1:

print(**"两次密码不一致"**)

**continue**

msg = **"R %s %s"** % (name, passwd)

*# 　发送请求*

s.send(msg.encode())

*# 　接收反馈*

data = s.recv(128).decode()

**if** data == **'OK'**:

print(**"注册成功"**)

login(name)

**else**:

print(**"注册失败"**)

**return**

*# 处理登录*

**def** do\_login():

name = input(**"User:"**)

passwd = getpass()

msg = **"L %s %s"** % (name, passwd)

s.send(msg.encode())

*# 　等待反馈*

data = s.recv(128).decode()

**if** data == **'OK'**:

print(**"登录成功"**)

login(name)

**else**:

print(**"登录失败"**)

*# 创建网络链接*

**def** main():

**while True**:

print(**"""**

**================Welcome===============**

**1. 注册 　2. 登录** **3. 退出**

**======================================**

**"""**)

cmd = input(**"输入选项:"**)

**if** cmd == **'1'**:

do\_register()

**elif** cmd == **'2'**:

do\_login()

**elif** cmd == **'3'**:

s.send(**b'E'**)

print(**"谢谢使用"**)

**return**

**else**:

print(**"请输入正确命令!"**)

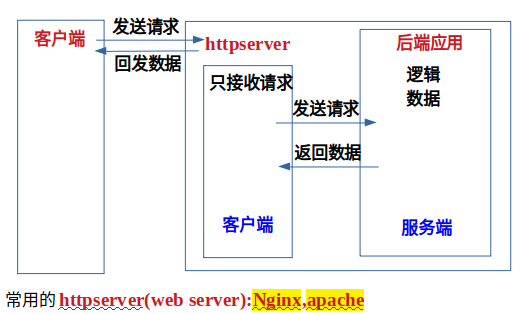
**if** \_\_name\_\_ == **"\_\_main\_\_"**:

main()

## HTTPServer３.０项目

参考代码：HTTPServer

* + 1. **图示：**



### **功能 ：**

httpserver部分

获取http请求  
解析http请求  
将请求发送给WebFrame  
从WebFrame接收反馈数据  
将数据组织为Response格式发送给客户端

WebFrame部分

从httpserver接收具体请求  
根据请求进行逻辑处理和数据处理  
将需要的数据反馈给httpserver

特点

采用httpserver和应用处理分离的模式,降低了耦合度  
采用了用户配置文件的思路  
webframe部分采用了模拟后端框架的处理方法

技术点

httpserver部分需要与两端建立通信  
webFrame部分采用多路复用接收并发请求  
数据传递使用json格式

项目结构：

|--httpserver –1.HttpServer.py (主程序)

| --2.config (httpserver配置)

project--|

|

|

|--WebFrame --1.WebFrame.py (主程序代码)

--2.static （存放静态网页）

--3.views.py （ 应用处理程序）

--4.urls.py （存放路由）

--5.settings （框架配置）

交互数据格式协议

httpserver--》webframe {method:'GET',info:'/'}

webframe-->httpserver {status:'200',data:'ccccc'}

### 代码实现：

#### **httpserver** 代码：

##### **１．HttpServer(主程序)：**

*"""*

*Aid httpserver v3.0*

*"""*

**from** socket **import** \*

**import** sys

**from** threading **import** Thread

**from** config **import** \*

**import** re

**import** json

*# 　服务器地址*

ADDR = (HOST, PORT)

*# 　和WebFrame通信*

**def** connect\_frame(env):

s = socket()

**try**:

s.connect((frame\_ip, frame\_port)) *# 连接ｗｅｂｆｒａｍｅ*

**except** Exception **as** e:

print(e)

**return**

*# 　将请求字典转换为ｊｓｏｎ数据发送*

data = json.dumps(env)

s.send(data.encode())

*# 　接收ｗｅｂｆｒａｍｅ数据,接收ｊｓｏｎ*

data = s.recv(4096 \* 100).decode()

**return** json.loads(data) *# 返回数据字典*

*# 　封装ｈｔｔｐｓｅｒｖｅ基本功能*

**class** HTTPServer(object):

**def** \_\_init\_\_(self, address):

self.address = address

self.create\_socket()

self.bind()

*# 　创建套接字*

**def** create\_socket(self):

self.sockfd = socket()

self.sockfd.setsockopt(SOL\_SOCKET, \

SO\_REUSEADDR, DEBUG)

*# 　绑定地址*

**def** bind(self):

self.sockfd.bind(self.address)

self.ip = self.address[0]

self.port = self.address[1]

*# 　启动服务器*

**def** serve\_forever(self):

self.sockfd.listen(5)

print(**"Listen the port %d..."** % self.port)

**while True**:

connfd, addr = self.sockfd.accept()

print(**"Connect from"**, addr)

client = Thread(target=self.handle, \

args=(connfd,))

client.setDaemon(**True**)

client.start()

*# 　处理具体的客户端请求*

**def** handle(self, connfd):

request = connfd.recv(4096).decode()

pattern = **r"(?P<method>[A-Z]+)\s+(?P<info>/\S\*)"**

**try**:

env = re.match(pattern, request).groupdict()

**except**:

connfd.close()

**return**

**else**:

data = connect\_frame(env)

**if** data:

self.response(connfd, data)

*# 　讲数据整理为响应格式发送给浏览器*

**def** response(self, connfd, data):

*# data {'status':'200',data:'content'}*

**if** data[**'status'**] == **'200'**:

responseHeaders = **"HTTP/1.1 200 OK\r\n"**

responseHeaders += **"Content-Type:text/html\r\n"**

responseHeaders += **"\r\n"**

responseBody = data[**'data'**]

**elif** data[**'status'**] == **'404'**:

responseHeaders = **"HTTP/1.1 404 Not Found\r\n"**

responseHeaders += **"Content-Type:text/html\r\n"**

responseHeaders += **"\r\n"**

responseBody = data[**'data'**]

**elif** data[**'status'**] == **'500'**:

**pass**

*# 　将数据发送给浏览器*

response\_data = responseHeaders + responseBody

connfd.send(response\_data.encode())

**if** \_\_name\_\_ == **"\_\_main\_\_"**:

httpd = HTTPServer(ADDR)

httpd.serve\_forever()

##### **２．config (HttpServer配置文件)：**

*# 　htttserver相关配置*

*# 　[http server ip]*

HOST = **'0.0.0.0'**

PORT = 8000

*# 　[debug]*

DEBUG = **True**

*# 　[frame ip]*

frame\_ip = **"127.0.0.1"**

frame\_port = 8080

#### **WebFrame** 代码：

##### **１．WebFrame (主程序代码)：**

*"""*

*模拟网站后端应用程序工作*

*"""*

**from** socket **import** \*

**import** json

**from** settings **import** \* *# 配置模块*

**from** select **import** select

**from** urls **import** \* *# urls模块*

frame\_address = (frame\_ip, frame\_port)

*# 　应用类,将功能封装在类中*

**class** Application(object):

**def** \_\_init\_\_(self):

self.sockfd = socket()

self.sockfd.setsockopt(SOL\_SOCKET, \

SO\_REUSEADDR, DEBUG)

self.sockfd.bind(frame\_address)

**def** start(self):

self.sockfd.listen(5)

print(**"Listen the port %d"** % frame\_port)

rlist = [self.sockfd]

wlist = []

xlist = []

*# 　select IO多路复用监听请求*

**while True**:

rs, ws, xs = select(rlist, wlist, xlist)

**for** r **in** rs:

**if** r **is** self.sockfd:

connfd, addr = r.accept()

rlist.append(connfd)

**else**:

self.handle(r)

rlist.remove(r)

**def** handle(self, connfd):

request = connfd.recv(1024).decode()

request = json.loads(request) *# 请求字典*

*# 　request==>{'method':'GET','info':'/'}*

**if** request[**'method'**] == **'GET'**:

**if** request[**'info'**] == **'/' or** \

request[**'info'**][-5:] == **'.html'**:

response = self.get\_html(request[**'info'**])

**else**:

response = self.get\_data(request[**'info'**])

**elif** request[**'method'**] == **'POST'**:

**pass**

*# 将数据发送给ｈｔｔｐｓｅｒｖｅｒ*

response = json.dumps(response)

connfd.send(response.encode())

connfd.close()

*# 　处理网页*

**def** get\_html(self, info):

**if** info == **'/'**:

filename = STATIC\_DIR + **"/index.html"**

**else**:

filename = STATIC\_DIR + info

**try**:

fd = open(filename)

**except** Exception **as** e:

f = open(STATIC\_DIR + **"/404.html"**)

**return** {**'status'**: **'404'**, **'data'**: f.read()}

**else**:

**return** {**'status'**: **'200'**, **'data'**: fd.read()}

**def** get\_data(self, info):

**for** url, func **in** urls:

**if** url == info:

**return** {**'status'**: **'200'**, **'data'**: func()}

**return** {**'status'**: **'404'**, **'data'**: **"Sorry...."**}

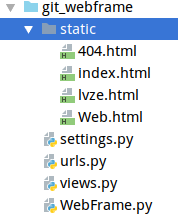
**if** \_\_name\_\_ == **"\_\_main\_\_"**:

app = Application()

app.start()

##### **２．static （存放静态网页文件夹）**

**文件需要如下：**



##### **３．views（ 应用处理程序）**

**import** time

**def** show\_time():

**return** time.ctime()

**def** say\_hello():

**return "Hello World"**

**def** say\_bye():

**return "Good bye!!"**

##### **４．urls （存放路由）**

**from** views **import** \*

urls = [

(**'/time'**,show\_time),

(**"/hello"**,say\_hello),

(**'/bye'**,say\_bye)

]

##### **５． settings （框架配置）**

*#　web frame 配置文件*

*# [frame ip]*

frame\_ip = **'127.0.0.1'**

frame\_port = 8080

*#　[debug]*

DEBUG = **True**

*#　[save html pages]*

STATIC\_DIR = **"./static"**