

- 准备工作
 - [Node.js](#)
 - [Git](#)
- 本地搭建Hexo博客环境
 - [初始化Hexo](#)
 - [配置Hexo](#)
 - [主题推荐](#)
 - [Butterfly](#)
 - [适用于Butterfly的繁体转换脚本](#)
 - [Matery](#)
 - [NexT](#)
 - [推送Hexo博客到GitHub](#)
 - [新建GitHub库](#)
 - [推送到GitHub](#)
- 写博客
 - [新建博文](#)
 - [修改模板](#)
 - [草稿文件夹](#)
 - [插入图片](#)
- 使用Gitee+PicGo搭建图床
 - [Gitee](#)
 - [新建图床仓库](#)
 - [新建Public仓库](#)
 - [修改分支名\(可选项\)](#)
 - [设置私人令牌](#)
 - [PicGo](#)
 - [安装PicGo](#)
 - [安装picgo-plugin-github-plus插件](#)
 - [Typora](#)
- [Python脚本在deploy前处理文章](#)
 - [脚本功能及内容](#)
 - [使用方法](#)

- Hexo description: LeoK77杂谈/使用Hexo+GitHubPages搭建个人博客 cover: /img/Hexo_Logo.svg date: 2021-02-09 09:53:38 updated: 2022-01-04 11:19:09

Hexo是一个快速、简洁且高效的博客框架。Hexo使用Markdown(或其他渲染引擎)解析文章，在几秒内，即可利用靓丽的主题生成静态网页。搭配GitHub Pages服务可以获得专属的个人博客。

以下内容以Windows系统的操作为例：

准备工作

Node.js

安装Node.js可以直接安装本体，也可以使用nvs、nvm等工具进行多版本管理，这里我使用的是Hexo推荐的nvs进行版本管理。

nvs的安装请参照其官网教程，这里只给出使用方法。

nvs官网：<https://GitHub.com/jasongin/nvs>

```
nvs          #如果有输出则说明已经可用，按ESC退出
#如果裸连node的官方库速度不慢，那么就不添加华为镜像源源
nvs remote node-huawei https://repo.huaweicloud.com/nodejs/ #添加华为云镜像源到nvs中
nvs          #运行nvs，选择“Download another version”，找到你刚才添加的“node-huawei”，然后选择版本
#推荐安装Node12的最新版，因为使用Node14的时候可能搭配一些主题会报Warning，虽然影响不大，但是看着心里不舒服
nvs ls       #列出安装的版本
#找到你刚才安装的node12版本，复制完整名字(我这里以在华为源下载的Node12为例)
nvs link node-huawei/12.20.1/x64 #将Node12设置为默认版本
nvs use node-huawei/12.20.1/x64  #启用Node12
```

设置Node.js12为默认环境：

```
Windows PowerShell
PS C:\Users\LeoK77> nvs ls
node-huawei/14.15.4/x64 (Fermium)
> node-huawei/12.20.1/x64 (Erbium)
PS C:\Users\LeoK77> nvs link node-huawei/12
$env:LOCALAPPDATA\nvs\default -> $env:LOCALAPPDATA\nvs\node-huawei\12.20.1\x64
User profile PATH += $env:LOCALAPPDATA\nvs\default
PS C:\Users\LeoK77> nvs use node-huawei/12
PS C:\Users\LeoK77> nvs ls
node-huawei/14.15.4/x64 (Fermium)
> #node-huawei/12.20.1/x64 (Erbium)
PS C:\Users\LeoK77> nvs

-----
| Select a version                                     |
+-----+
| a) node-huawei/14.15.4/x64 (Fermium)                 |
| [b] node-huawei/12.20.1/x64 (Erbium) [current] [default] |
| ) Download another version                          |
| .) Don't use any version                            |
+-----+

Type a hotkey or use Down/Up arrows then Enter to choose an item.
```

完整名: node-huawei/12.20.1/x64
从node-huawei只下载了一个Node12
只需要用缩写 node-huawei/12
current: 正在用
default: 默认

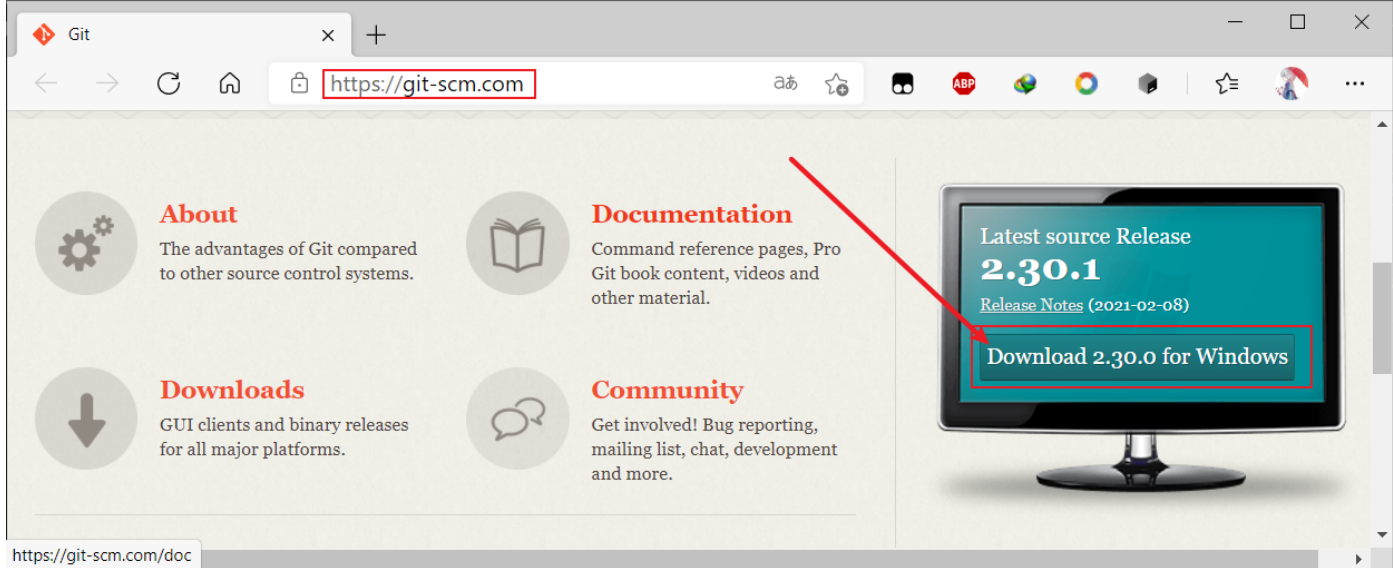
“node/14.15.4/x64 (Fermium)”运行Hexo(使用主题Butterfly)时报的warning如下:

```
INFO Start processing
INFO Hexo is running at http://localhost:4000 . Press Ctrl+C to stop.
(node:8492) Warning: Accessing non-existent property 'lineno' of module exports inside circular dependency
(Use 'node --trace-warnings ...' to show where the warning was created)
(node:8492) Warning: Accessing non-existent property 'column' of module exports inside circular dependency
(node:8492) Warning: Accessing non-existent property 'filename' of module exports inside circular dependency
(node:8492) Warning: Accessing non-existent property 'lineno' of module exports inside circular dependency
(node:8492) Warning: Accessing non-existent property 'column' of module exports inside circular dependency
(node:8492) Warning: Accessing non-existent property 'filename' of module exports inside circular dependency
```

```
npm config set registry https://repo.huaweicloud.com/repository/npm/ #设置为华为云仓库镜像源
npm config list #查看设置是否成功
npm i hexo-cli -g #全局安装hexo
```

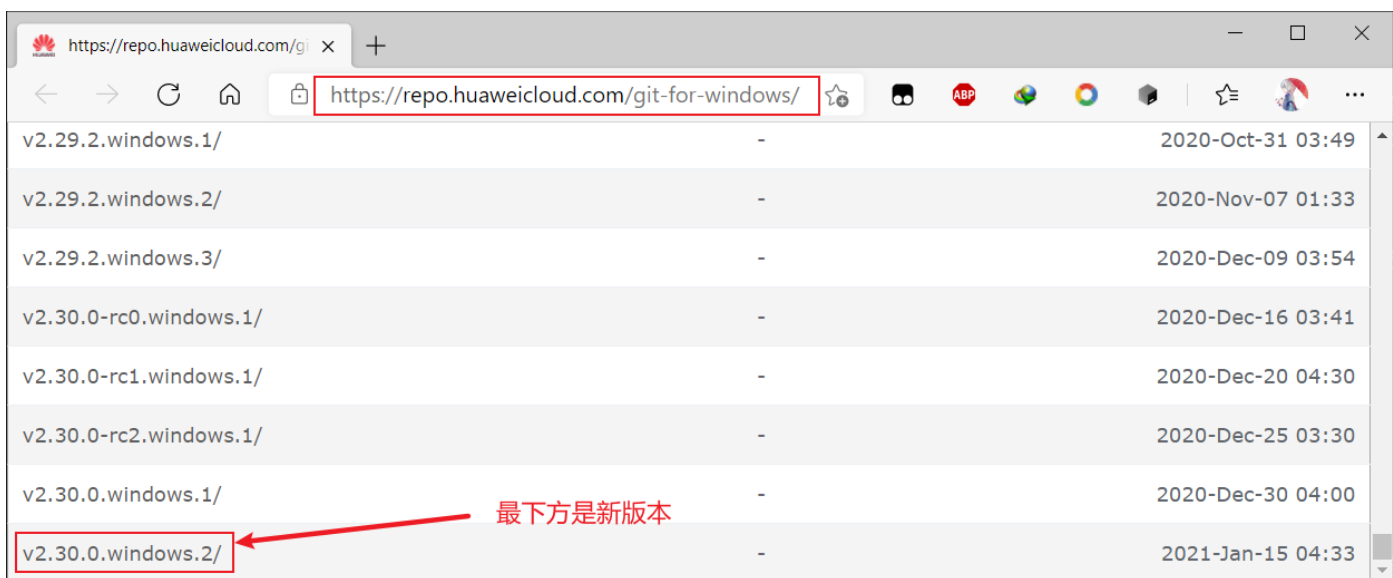
Git

Git官网: <https://git-scm.com/>



如果访问官网的下载速度过慢，可以从各大镜像网站下载，比如华为开源镜像站：

华为开源镜像站 Git for Windows : <https://repo.huaweicloud.com/git-for-windows/>



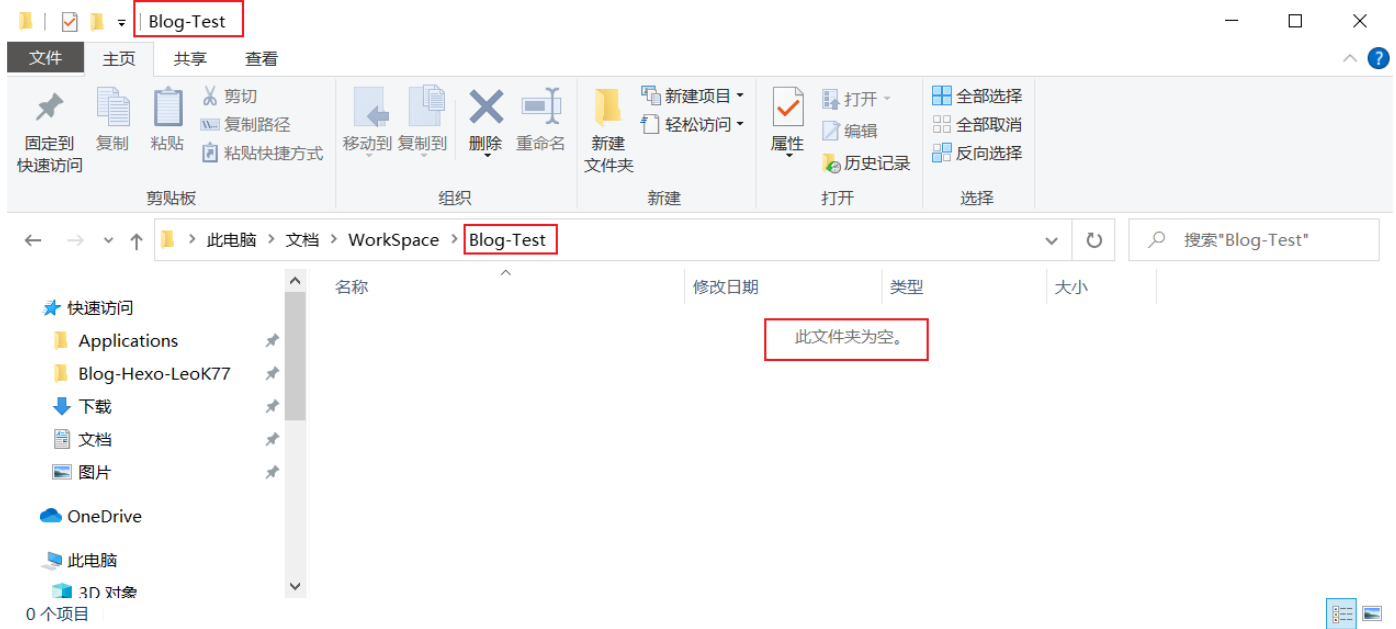
安装步骤不做演示。

本地搭建Hexo博客环境

初始化Hexo

新建一个文件夹来作为你的本地Hexo博客，你的所有Hexo博客内容都应该存放在这个文件夹中。

比如我新建了一个名为“Hexo-Test”的文件夹，那么我的所有博客内容都在这个文件夹里。



打开Windows终端，通过“cd”指令到达这个文件夹；或者在这个文件夹中摁着SHIFT再右键，选择“在此处打开PowerShell窗口”。

```
hexo init #初始化Hexo博客，这个文件夹需要是空文件夹
```

```
Windows PowerShell
PS C:\Users\LeoK77\Documents\Workspace\Blog-Test> hexo init
INFO Cloning hexo-starter https://github.com/hexojs/hexo-starter.git
132mINFO 139m Install dependencies
added 188 packages from 443 contributors in 4.016s

15 packages are looking for funding
  run `npm fund` for details

INFO Start blogging with Hexo!
PS C:\Users\LeoK77\Documents\Workspace\Blog-Test>
```

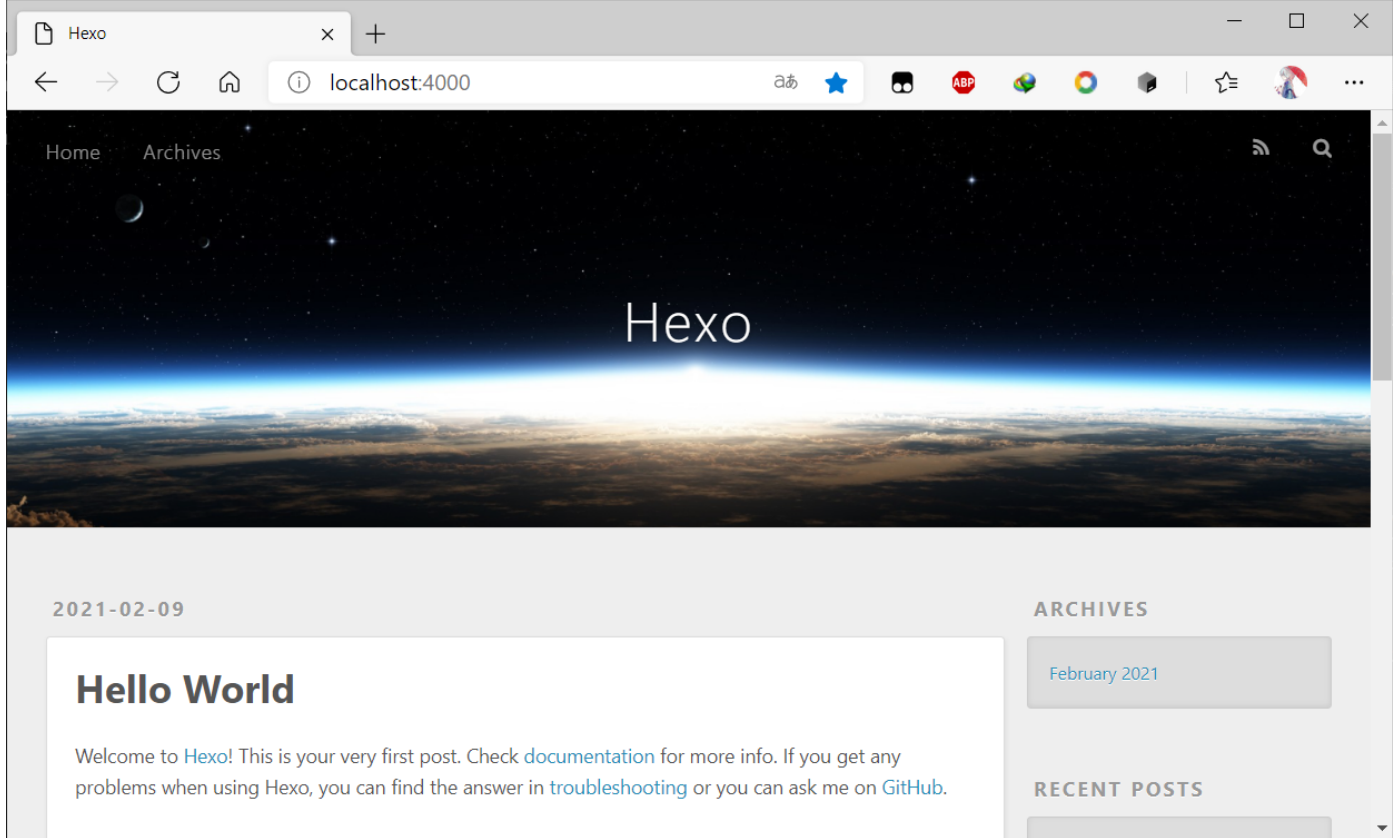
刚才创建的空文件夹位置

如果这里报错，有两种情况：
一、这个文件夹不是空文件夹——删除文件夹中内容
二、没有全局安装hexo-cli——执行npm i hexo-cli -g

```
npm install # 安装依赖
hexo s     # 本地运行hexo博客
```

```
Windows PowerShell  hexo
PS C:\Users\LeoK77\Documents\Workspace\Blog-Test> hexo s
INFO Validating config
INFO Start processing
INFO Hexo is running at http://localhost:4000. Press Ctrl+C to stop.
```

如图所示，本地访问端口是<http://localhost:4000>，运行时在浏览器打开这个网址，就可以查看Hexo博客了。



配置Hexo

以下仅列出几个我认为有必要改的_config.yml文件中的内容：

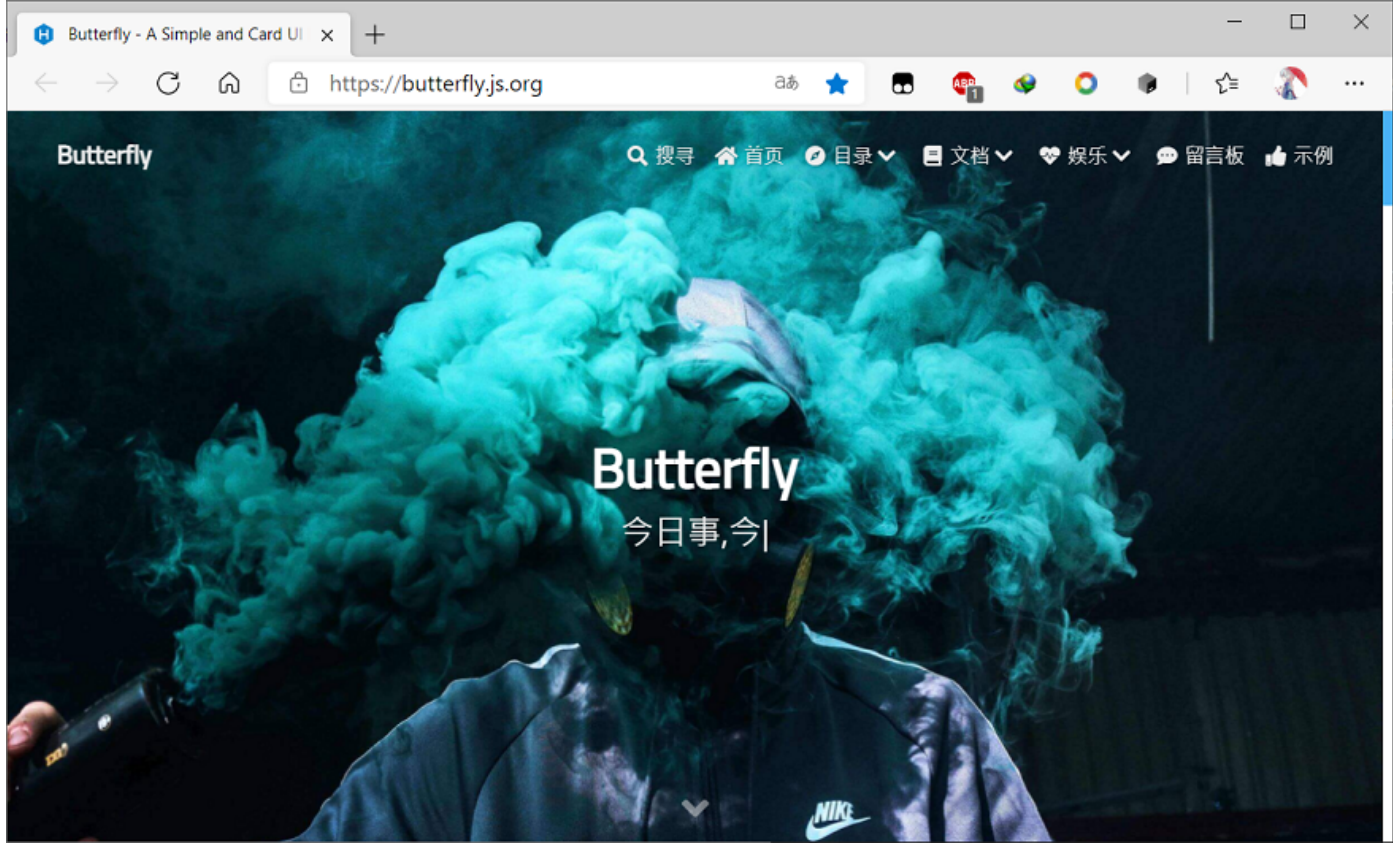
```
# Site
title: Hexo          #博客标题
author: John Doe    #博客作者，改成自己名字
language: en        #博客语言，中文是 zh-CN
# URL
url: https://leok77.github.io  #你要发布到哪儿，比如我用GitHubPages，就是这个网址，把
“leok77”修改为你的用户名即可
# THEME
theme: landscape    #这个是默认主题，可以自己换个好看/高效的主题
```

主题推荐

Butterfly

GitHub: <https://github.com/jerryc127/hexo-theme-butterfly> 主题首页：
<https://butterfly.js.org/>

美化效果很好的一个主题，功能集成也比较完善，我现在使用的就是这个主题(然而我追求简洁又懒，所以把绚丽的动画效果都关了)，主题首页有相应的配置教程，真的是“手把手”教你怎么把这个主题配置好，总之就是非常棒。



我在使用的是将配置文件另存到了博客根目录，这样在升级主题的时候就不需要担心自己的配置文件被默认文件覆盖了。

名称	修改日期	类型
.deploy_git	2021/3/20 22:59	文件夹
node_modules	2021/3/20 13:35	文件夹
public	2021/3/20 22:59	文件夹
scaffolds	2021/3/12 16:56	文件夹
source	2021/3/12 21:29	文件夹
themes	2021/3/3 10:34	文件夹
gitignore	2021/3/1 11:57	文本文档
<input checked="" type="checkbox"/> ! _config.butterfly	2021/3/20 13:53	Yaml 源文件
! _config.landscape	2021/3/1 11:57	Yaml 源文件
! _config	2021/3/20 13:45	Yaml 源文件
db	2021/3/20 22:59	JSON 源文件
deploy	2021/3/1 11:57	Windows 批处理...
package	2021/3/20 13:35	JSON 源文件
package-lock	2021/3/20 13:35	JSON 源文件

将"博客文件夹/node_modules/hexo-theme-butterfly"下的
_config.yml另存到"博客文件夹"根目录下并重命名为
_config.butterfly.yml

适用于**Butterfly**的繁体转换脚本

因为**Butterfly**的配置文件默认是繁体字，而我使用简体字，所以我需要将其转换为简体字之后才可以更好的阅读和使用，这里用的是开源的openCC库

```

from opencv import OpenCC # OpenCC开源简繁转换
import os.path
import path_analysis

def tradition_to_simple(src_path: str):
    abspath_src = os.path.abspath(src_path)
    if not os.path.exists(abspath_src):
        print('ERROR! FILE ** ' + abspath_src + ' ** NOT EXIST')
        return
    abspath_dst = path_analysis.get_abspath_dst(abspath_src)
    converter = OpenCC('t2s.json')
    with open(abspath_dst, 'w', encoding='UTF-8') as dst:
        with open(abspath_src, 'r', encoding='UTF-8') as src:
            for sentence in src:
                dst.write(converter.convert(sentence))

```



```
path_analysis.replace_or_not(abspath_src, abspath_dst)
```

```
if __name__ == '__main__':  
    # 使用时将这个路劲替换为自己的相应路径即可  
    tradition_to_simple(  
        r'C:\Users\LeoK77\Documents\WorkSpace\Blog-Hexo-LeoK77\node_modules\hexo-  
theme-butterfly\_config.yml'  
    )
```

Matery

GitHub: <https://github.com/blinkfox/hexo-theme-matery> 主题首页:
<https://blinkfox.github.io/>

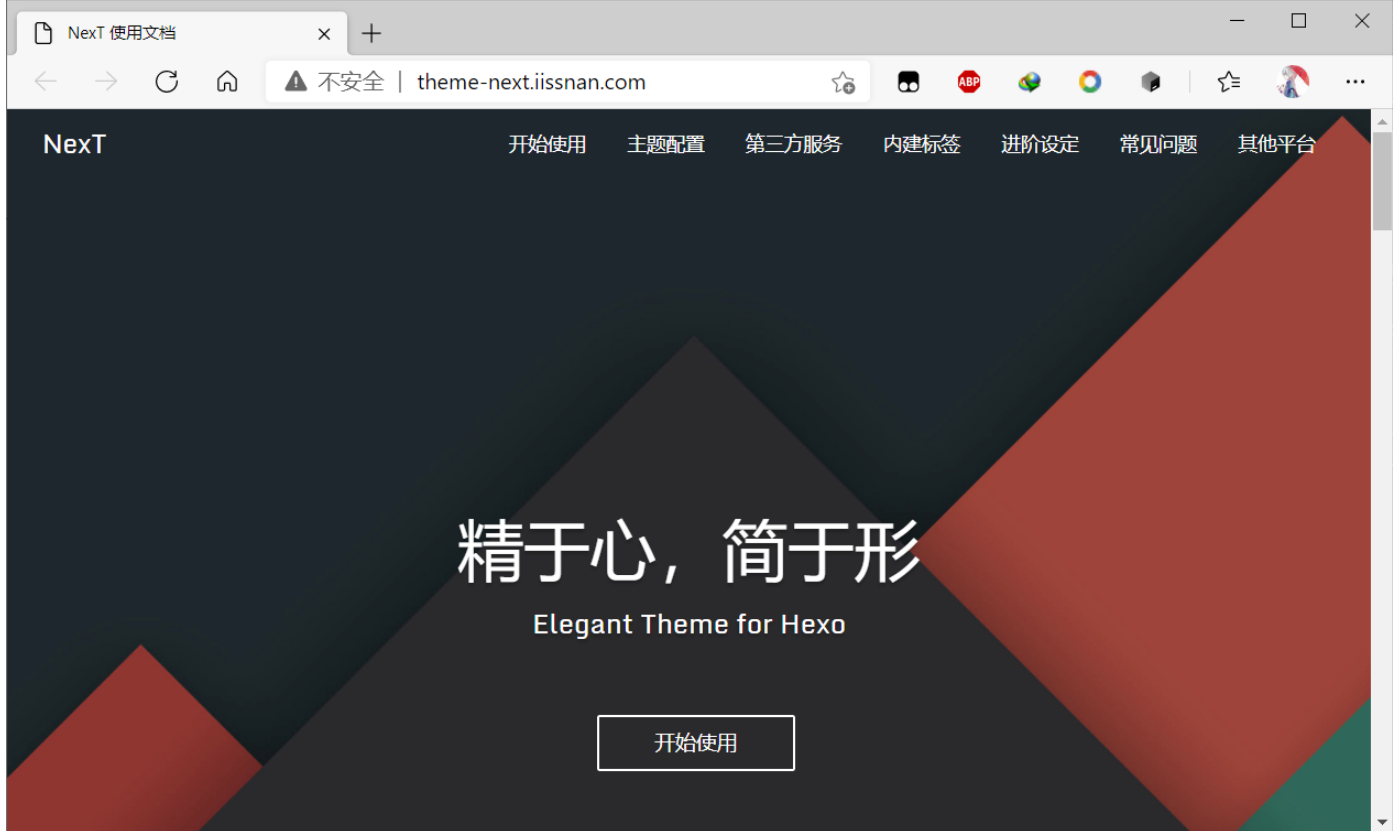
美化效果好，有较为详细的教程，但是我没弄明白那个头部的“彩虹换色”怎么关，导致我放的壁纸在动态换色下显得特别呆，所以就没继续用这个主题。



NexT

GitHub: <https://github.com/theme-next/hexo-theme-next> 主题旧版首页:
<http://theme-next.iissnan.com/> 主题首页: <https://theme-next.org/>

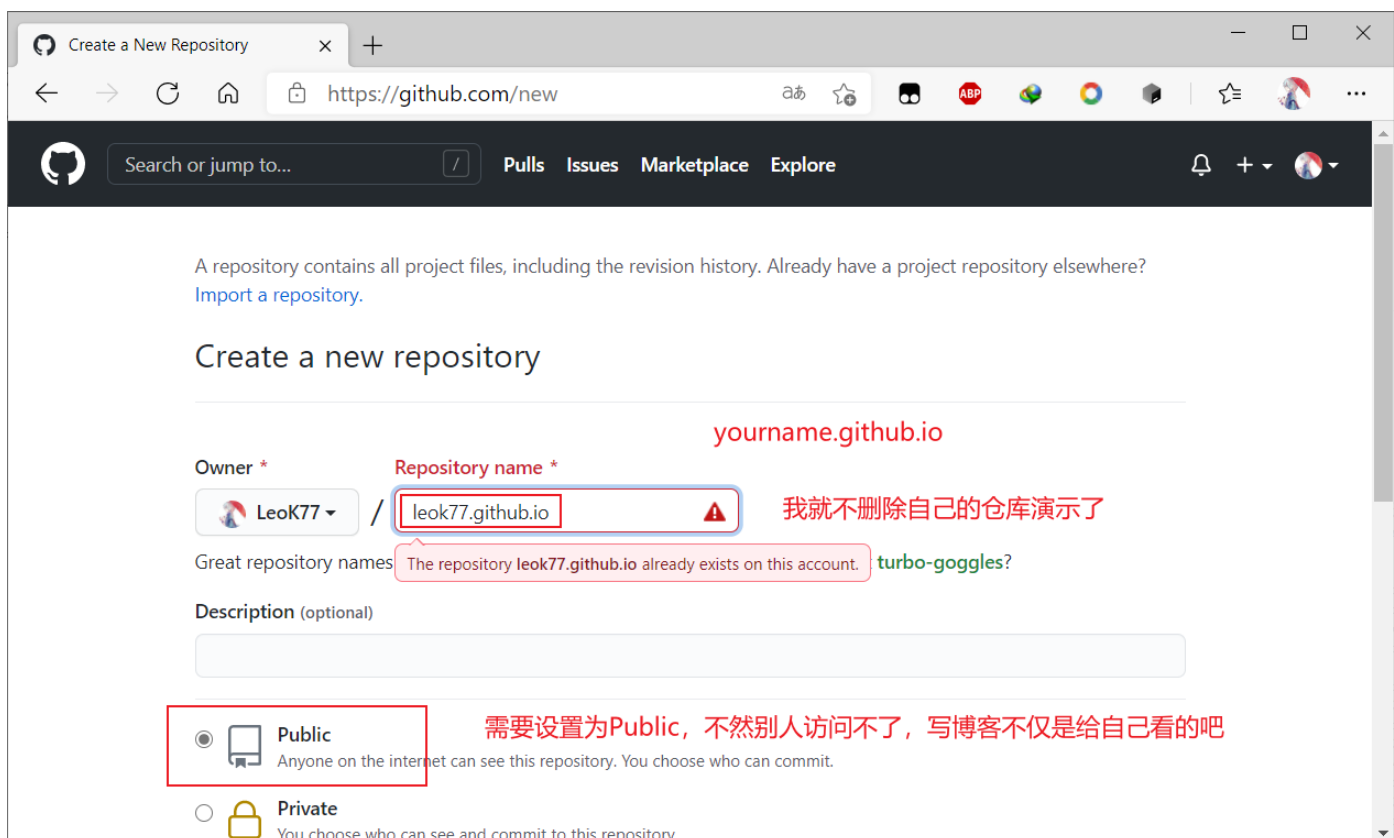
如果你去搜Hexo的教程或者视频，那么他们推荐的主题很可能就是NexT，这是一个十分简约的主题，但是提供极多的插件，这也就意味着高度定制性，你只需要安装你想要的功能，无论是简约还是绚丽美化，NexT都可以做到。



推送Hexo博客到GitHub

新建GitHub库

需要在自己的GitHub中新建一个“username.github.io”的public仓库，比如我的用户名是leok77，那么我的仓库是“leok77.github.io”。



推送到GitHub

```
npm i hexo-deployer-git --save    #安装hexo-deployer-git插件
```

配置_config.yml文件

```
# Deployment
## Docs: https://hexo.io/docs/one-command-deployment
deploy:
  - type: git      #方式
    repo: https://github.com/username/username.github.io.git
    branch: main   #分支 - 都呼吁分支名改为 main
#同时推送到多个仓库
  - type: git
    repo: https://gitee.com/username/username.git
    branch: main
```

推送到GitHub:

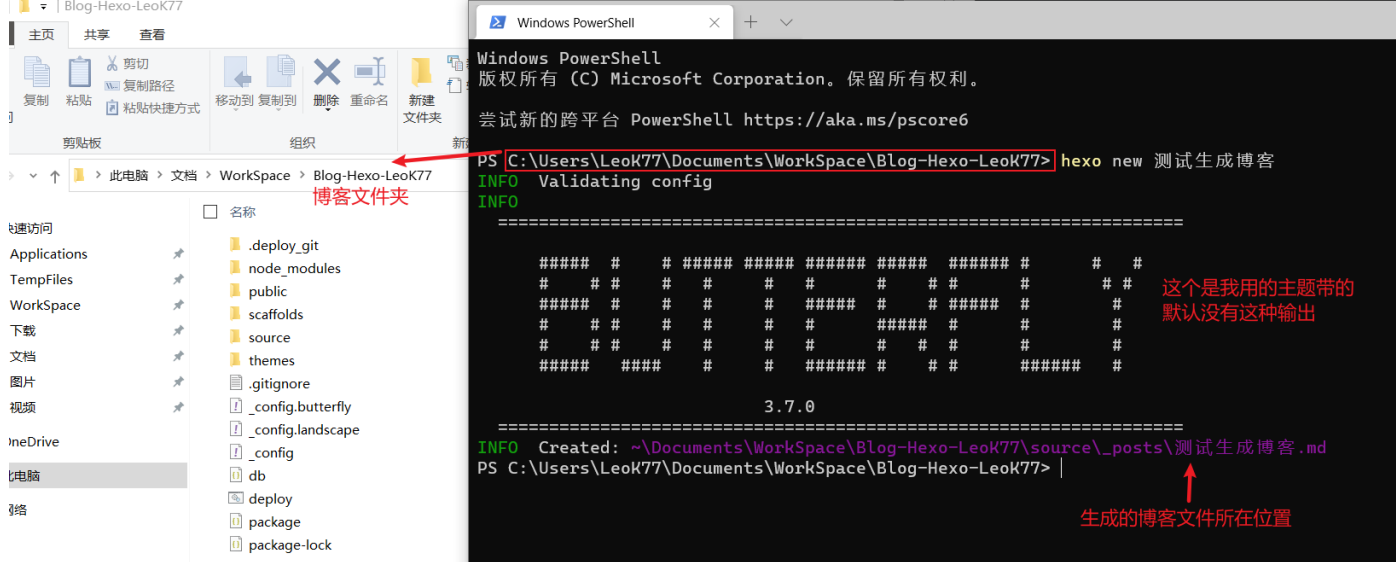
```
hexo clean    #清除旧内容
hexo g        #创建新网页
hexo d        #推送到GitHub
```

写博客

新建博文

在Hexo博客文件夹下打开命令行，执行"**hexo new title**"命令，其中**title**是你想起的博客名字。

```
hexo new title
```



然后根据输出提示的博文地址前去打开编辑即可，我使用的Markdown编辑器是**VSCode**和**Typora**；至于Markdown语法我就不赘述了，一搜一大把。

修改模板

如果你希望自己的博文文件在生成时就带着自己自定义的内容，那么可以通过修改默认模板文件来解决。

进入"博客文件夹/scaffolds"目录，先将"post.md"文件备份一下，比如我这里将它另存为了"post_default"，以后想退回到默认post文件时可以直接替换。

下面是我的模板文件，相比源模板增加的内容并不多，但是较为实用：

- **title**: 我在创建文章的时候**title**其实是**categories/title**
- **author**: 作者名字
- **categories**: 分类 - 设置为**default**是为了这一栏不会因为是空的而被放到最后
- **tags**: 标签 - 设置为**default**同理
- **description**: 描述，这里我设置为 **categories/title**，与我**gitee**的图床仓库地址对应
- **cover**: 这个是**Butterfly**主题带的封面图片
- **date**: 创建日期
- **updated**: 更新日期，如果不设置更新日期的话，那么之后更换设备的时候，文章更新日期就会全变成最新日期了

```
title: {{ title }}
author: LeoK77
categories: default
tags:
  - default
description: {{ title }}
date: {{ date }}
updated: 2022-01-04 11:19:09
```

设置后再使用hexo new的时候，就不用自己手动添加这些内容了，依次修改对应项即可。

草稿文件夹

是否存在有的博文先写好而有的博文还没写完的情况？而又不想等到最后一起发布？那么这时候就需要草稿文件夹了。

在"博客文件夹/source/"目录下新建"_drafts"文件夹，将还不想发布的放到这个文件夹内即可，这样在推送博文的时候只需要控制好哪些在"_posts"(要发布)和哪些在"_drafts"(草稿文件夹，不发布)即可。

如果想使用命令进行控制的话，可以参见[hexo文档](#)，但我习惯自己手动换了(之前用hexo-admin插件替换控制草稿的时候，我的博文文件总是会第一行被它干掉，搞的我很不爽，就不用命令了，还是自己手动换不会出错)。

插入图片

如果想使用资源文件夹(相对路径，图片存储在GitHub上)的话，可以参见[Hexo文档](#)，我这里也不赘述了。但是需要注意的是图片存储在GitHub的情况下可能会导致国内访问速度特别慢(至少我这里是这样)，我就是因为图片加载特别慢才使用Gitee当免费图床，有预算的话可以考虑下七牛云、阿里云、腾讯云等对象云存储服务，可惜我知道的时候已经不能白嫖了，所以我选择白嫖Gitee(手动狗头)。

使用Gitee+PicoGo搭建图床

虽然Hexo可以设置相对路径引用存储图片，但是这样会把图片存储到Github上，而碍于国内对于Github的访问速度，博客中的图片加载起来十分的慢，而穷孩子用不起对象云存储，小型图床又怕跑路，所以把主意打到了Gitee身上

Gitee

如果没有Gitee账号的话自行注册一个，这里就不演示账号注册了。

Gitee官网: <https://gitee.com/>

新建图床仓库

新建Public仓库

在Gitee主页左下角可以找到新建仓库按钮，点击之后即可新建一个开源(public)仓库，因为要制作的Hexo博客图床是希望所有人访问你的博客的时候都可以看到图片，如果设置为私有仓库别人就无法访问图片了。



建议勾选“使用Readme文件初始化这个仓库”，因为PicGo在上传图片的时候需要Gitee中已经有相应的分支了，如果不初始化仓库，那么默认的“master”分支是不存在的。

新建仓库

仓库名称 ✓

blog-img-test

仓库名称随便填，符合规范即可

归属

LeoK77

路径 ✓

blog-img-test

仓库地址: <https://gitee.com/leok77/blog-img-test>

仓库介绍 非必填

用简短的语言来描述一下吧

设置为开源，这样其他人才可以访问你的仓库/图床

是否开源

☐ 私有

☒ 公开

任何人都可以访问该仓库的代码和其他任何形式的资源

选择语言

请选择语言

添加 .gitignore

请选择 .gitignore 模板

添加开源许可证 ⓘ

请选择开源许可证

点此快速选择许可证

☒ 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 ⓘ

直接先初始化这个仓库，
这样就不用手动初始化了

选择分支模型 (仓库初始化后将根据所选分支模型创建分支)

单分支模型 (只创建 master 分支)

[导入已有仓库](#)

创建

修改分支名(可选项)

如果你没有把仓库分支名更改为“main”的习惯，那么“新建图床仓库”这一部分就已经做完了，如果你希望你的仓库分支名是“main”，那么请继续操作。

仓库新建完毕之后会自动转到仓库页面，在这里点击如图所示位置的“分支”。



在新弹出的分支设置窗口点击右侧的“新建分支”，分支名填“main”，然后将“main”分支设置为默认分支并删除“master”分支。



设置私人令牌

点击网页右上角的头像，然后点击弹出的“设置”



下滑找到左侧的安全设置，并选择“私人令牌”



点击网页右上角的“新建私人令牌”，然后设置私人令牌的描述及权限，权限只给如图所示的这两个就可以了。

私人令牌描述

blog-img-test

随便添加一句描述

让自己知道这个令牌干啥的就行

请选择将要生成的私人令牌所拥有的权限

☐ 全选

☒ user_info

访问你的个人信息、最新动态等

☒ projects

查看、创建、更新你的项目

☐ pull_requests

查看、发布、更新你的 Pull Request

☐ issues

查看、发布、更新你的 Issue

☐ notes

查看、发布、管理你在项目、代码片段中的评论

☐ keys

查看、部署、删除你的公钥

☐ hook

查看、部署、更新你的 Webhook

☐ groups

查看、管理你的组织以及成员

☐ gists

查看、删除、更新你的代码片段

☐ enterprises

查看、管理你的企业以及成员

☐ emails

查看你的个人邮箱信息

仅选这两项即可

提交

取消

如图所示区域就是你的私人令牌，使用PicGo上传图片到Gitee仓库中需要使用，此页面关闭后，Gitee就不会再显示私人令牌，自己复制下来保存好。



你的私人令牌blog-img-test已生成

[Blurred private token]

复制

本页面关闭后，平台将不再显示私人令牌，请妥善保存。

☒ 我已经了解个人令牌不再明文显示在平台上，并且已经复制保存好该令牌。

确认并关闭














PicGo

PicGo官网: <https://molunerfinn.com/PicGo/> PicGo Github:

<https://github.com/Molunerfinn/PicGo>

安装PicGo

无论是从官网下载PicGo还是从Github主页进去下载，最后都是要在Github上下载PicGo的Release，我此处下载的是“2.2.2”版本，并不是因为我2，而是因为这个是最新的稳定版，首页的“2.3.0-beta.4”是beta版，虽然功能更强大，但是beta不稳定啊，万一出了问题挂了不就不好了(我下面之所以提供的脚本里包含url转换就是因为之前用beta的时候不知道为啥就自动转码了，换回正式版就没这个问题了)。

 latest-linux.yml	
 latest-mac.yml	
 latest.yml	
 PicGo-2.2.2-mac.zip	
 PicGo-2.2.2.AppImage	
 PicGo-2.2.2.dmg	
 PicGo-2.2.2.dmg.blockmap	
 PicGo-Setup-2.2.2.exe	 win10
 PicGo-Setup-2.2.2.exe.blockmap	
 picgo_2.2.2_amd64.snap	
 Source code (zip)	
 Source code (tar.gz)	

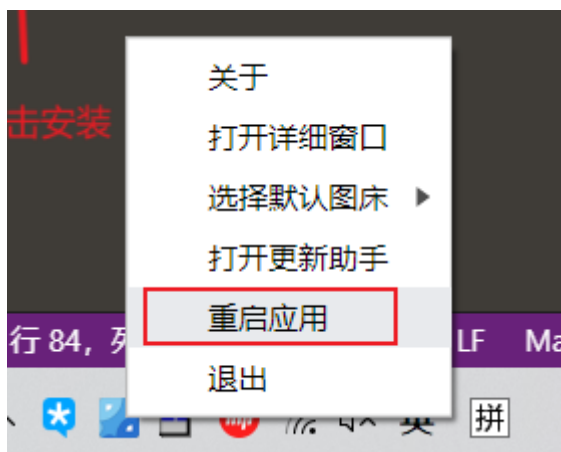
下载下来之后安装即可，不做演示。

安装picgo-plugin-github-plus插件

打开PicGo主面板，按照如图所示顺序安装“github-plus”插件，写作github-plus，但是也可以关联gitee，毕竟某种意义上gitee就是中国的github。



安装成功之后重启应用。在系统托盘区域，右键PicGo的图标，然后点击重启应用，重启后他是默认在托盘区域的，需要手动点一下，然后就出来了。



设置GithubPlus图床：

```
repo = username/repository_name #用户名/仓库名
branch = main #分支名字，如果前面没设置，那么这里用master即可
token = 私人令牌 #你前面设置的Gitee的私人令牌
path = #图片存储路径，如果不设置的话就会存储到主目录，建议分类
origin = gitee #选择gitee才会传到gitee
```

如图所示区域就是我的“repo”——“leok77/blog-img-test”



设置path的好处就是图片存储的位置在文件夹里，而不是全都存在主目录，这样当自己手贱同一张图片多次上传，或者要修改某一篇博文的内容的时候，可以通过找对应文件夹，而不是看着全部的主目录两眼一抹黑。



我设置里的repo和我前面提的repo不完全一致是因为这里设置中的是我一在用的图床仓库，前面那是我为了演示新建的仓库，懒得换了(懒癌晚期)，记得将“GithubPlus”设置为默认，并且点击“确认”，我就因为没点确认导致路径不生效从而图片存乱过(反面教材)。



Typora

Typora官网: <https://typora.io/>

我现在写博客是通过VSCode(我知道VSCode也有PicGo插件,但是我还没搞明白怎么让他传到Gitee上)+Typora的形式,博客中插入图片以及看Markdown都是通过Typora,因为Typora用来做阅读器十分的舒服,以及插入图片的时候可以配合PicGo直接上传,没有人想每次插入图片的时候,都打开PicGo——上传图片——等待连接——粘贴到博文中吧。



这样设置好之后,无论你是把图片直接拖动到Typora中,还是把Snipaste截图后的图片粘贴到Typora中,他都会自动帮你自动调用PicGo上传图片,并直接转化为可用的图片连接。

Python脚本在deploy前处理文章

脚本功能及内容

1. 文章内部的updated时间戳更新：我希望我的博客首页展示的是最近更新的文章而不是最近创建的文章，所以我选择hash比对的形式，每次推送前先SHA265处理每篇文章，然后将hash值改变的进行更新，避免了每次更新完文章之后都要手动修改时间戳
2. 图片连接的url转义：由于我在一段时间内使用了PicGo的某版beta版(换回beta版之后没有这个问题)，之前使用的时候生成的图片url默认就是中文，但是后来变成了特殊字符转置之后的，也就是我的图片连接变成了一串"%NN"串，虽然这样兼容的编辑器多了，但是影响可读性，索性写了个脚本转置回中文，写完博客后运行一下再发布，美滋滋。

```
import hashlib
import time
import datetime
import urllib.parse as parse
import os

def hash_check(filename: str) -> str:
    m = hashlib.sha256()
    abspath = os.path.abspath(filename)
    with open(abspath, 'rb') as src:
        while True:
            src_data = src.read(2048)
            if not src_data:
                break
            m.update(src_data)
    hash_result = m.hexdigest()
    # 返回hash_sum—全部大写的形式
    return hash_result.upper()

# 获得root_path目录下的所有文件路径
def get_all_path(root_dir_path: str) -> list:
    root_dir_path = os.path.abspath(root_dir_path)
    if not os.path.exists(root_dir_path):
        print('ERROR! ** ' + root_dir_path + ' ** NOT EXIST!!!')
        exit(0)
    all_file_path = []
    # 如果是 文件 或者是 空文件夹，直接添加到all_path中
    if os.path.isfile(root_dir_path) or (not os.listdir(root_dir_path)):
        all_file_path.append(root_dir_path)
    else:
        for root, dirs, files in os.walk(root_dir_path):
            for dir_name in dirs:
```

```

        dir_path = os.path.join(root, dir_name)
        if not os.listdir(dir_path):
            all_file_path.append(dir_path)
    for basename in files:
        all_file_path.append(os.path.join(root, basename))
return all_file_path

```

返回 abspath_dst , 在源文件名后面加日期

```

def get_abspath_dst(abspath_src: str) -> str:
    abspath_src = os.path.abspath(abspath_src)
    if not os.path.exists(abspath_src):
        print('ERROR! ** ' + abspath_src + ' ** NOT EXIST!')
        exit(0)
    path_without_ext, extended_name = os.path.splitext(abspath_src)
    abspath_dst = path_without_ext + datetime.datetime.now().strftime('%Y-%m-%d-%H-%M-%S') + extended_name
    return abspath_dst

```

是否用abspath_dst替代abspath_src

如果两者hash不一致, 则替代, 如果hash一致, 则不替代

```

def replace_or_not(abspath_src: str, abspath_dst: str):
    if hash_check(abspath_src) == hash_check(abspath_dst):
        os.remove(abspath_dst)
    else:
        os.replace(abspath_dst, abspath_src)

```

被转义的特殊字符转义回去

```

def url_escape_to_chinese(filename: str):
    abspath_src = os.path.abspath(filename)
    if not os.path.exists(abspath_src):
        print('ERROR! ** ' + abspath_src + ' ** NOT EXIST!')
        return
    abspath_dst = get_abspath_dst(abspath_src)
    with open(abspath_src, 'r', encoding='utf-8') as src_file:
        with open(abspath_dst, 'w', encoding='utf-8') as dst_file:
            for line_cur in src_file.readlines():
                if line_cur.find('http') != -1:
                    dst_file.write(parse.unquote(line_cur))
                else:
                    dst_file.write(line_cur)
    replace_or_not(abspath_src, abspath_dst)

```

如果hash结果与存储的结果不一致, 则更新时间并更新存储库里的hash值

```

def time_update(filename: str):
    abspath_src = os.path.abspath(filename)
    if not os.path.exists(abspath_src):
        print('ERROR! ** ' + abspath_src + ' ** NOT EXIST!')
        return
    abspath_dst = get_abspath_dst(abspath_src)
    with open(abspath_src, 'r', encoding='utf-8') as src_file:
        with open(abspath_dst, 'w', encoding='utf-8') as dst_file:
            for line_cur in src_file.readlines():
                if line_cur.find('updated:') == 0:
                    line_cur = line_cur.split()[0]
                    line_cur += ' '

```

```

        # 将更新时间设置为此文件的修改时间
        change_time = os.path.getmtime(abspath_src)
        change_time = time.strftime('%Y-%m-%d %H:%M:%S',
time.localtime(change_time))
        line_cur += change_time
        print(abspath_src + '----updated:----' + change_time)
        line_cur += '\n'
        dst_file.write(line_cur)
    replace_or_not(abspath_src, abspath_dst)

if __name__ == '__main__':
    root_path = r'C:/Users/LeoK77/Documents/WorkSpace/Blog-Hexo-LeoK77/source'
    all_path = get_all_path(root_path)
    hash_record_dict = {}
    hash_record_file = 'hash_record.txt'
    if not os.path.exists(hash_record_file):
        with open(hash_record_file, 'w', encoding='utf-8') as hash_config:
            for article_path in all_path:
                if article_path.find('.md') != -1:
                    hash_config.write(article_path + '-----' +
hash_check(article_path) + '\n')
            exit(0)
    with open(hash_record_file, 'r', encoding='UTF-8') as hash_record:
        for line in hash_record.readlines():
            # 当且仅当不是空行的时候才是存储hashRecord的行
            if len(line) != 0:
                list_tmp = list(line)
                list_tmp.pop() # 删除换行符
                record_cur = ''.join(list_tmp).split('-----')
                hash_record_dict[record_cur[0]] = record_cur[1]
    # url转义, 将%xx形式转义回原来的字符
    for article_path in all_path:
        if article_path.find('.md') != -1:
            url_escape_to_chinese(article_path)
    # 进行hash比较, 如果hash结果不匹配, 则更改时间并重新hash
    for article_path in all_path:
        if article_path.find('.md') == -1:
            continue
        record_exists = False
        if hash_record_dict.get(article_path, 'no_record') != 'no_record':
            record_exists = True
            if hash_record_dict[article_path] != hash_check(article_path):
                time_update(article_path)
                hash_record_dict[article_path] = hash_check(article_path)
        if not record_exists:
            hash_record_dict[article_path] = hash_check(article_path)
    with open(hash_record_file, 'w', encoding='utf-8') as hash_config:
        for key, val_hash in hash_record_dict.items():
            hash_config.write(key + '-----' + val_hash + '\n')

```

注：因为我本身就没想过要在gitee图床里带空格，所以我这里没考虑不转置空格的情况，而Markdown里一个内含空格的链接是会被断开的，所以如果有需求的话可以自己改一下

使用方法

使用也很简单，将上述python代码保存为一个python文件，然后写完博文之后python跑一下即可；我这里将跑的代码保存为了一个bat文件放在桌面上，这样我只需要双击它就可以了。

1. `cd "C:\Users\LeoK77\Documents\WorkSpace\PyCharm-Projects\ToolBox\"`

- 上述代码在我电脑里储存在这个位置，由于我使用的时候是两个文件，并不像上面结合在了一起，所以我需要cd 到这个路径再跑代码

2. `python.exe .\urlEscape_by_urllib.py`

- python运行即可

3. `pause`

- 这个是为了让窗口停顿一下，我好知道有没有出问题，不然窗口一闪而过，我不知道执行结果是好的还是坏的

```
cd "C:\Users\LeoK77\Documents\WorkSpace\PyCharm-Projects\ToolBox\"
python.exe .\urlEscape_by_urllib.py
pause
```

