# 文档分类向量化

计算机学院 07152001班

1120201198 史桠彬

**目录**

[文档分类向量化 1](#_Toc3714)

[1. 实验内容 2](#_Toc16118)

[2. 实验目的 2](#_Toc18050)

[3. 实验环境 3](#_Toc18)

[4. 实验步骤 3](#_Toc20988)

[4.1 搭建MPI运行环境 3](#_Toc21215)

[4.2 创建源代码 5](#_Toc21759)

[4.3 进行编译 11](#_Toc17972)

[4.4 运行程序 12](#_Toc14853)

[5. 实验分析 15](#_Toc15798)

[5.1 实验内容分析 15](#_Toc13544)

[5.2 实验结果分析 17](#_Toc10614)

[6. 问题与讨论 19](#_Toc23178)

[7. 实验心得 19](#_Toc897)

[参考资料 20](#_Toc23222)

## 1. 实验内容

当前各领域对使用计算机处理信息均有着巨大的需求。在许多应用领域（信息检索、情感分析、垃圾邮件过滤、文本分类等）中，我们希望实现对大量文本数据的自动处理分析，但基于机器语言的计算机无法直接理解和处理文本数据，因此，我们需要一种能将文本数据转换为计算机可处理的数值表示的技术，文档分类向量化应运而生。

文档分类向量化是将包含文本信息的文档表示为数值向量的过程。通过将文档转换为数值向量，计算机可以利用特征表示、向量空间模型等对文本进行分类及后续分析。

本实验参考清华大学出版社《MPI与OpenMP并行程序设计：C语言版》内容，主要基于**MPI**和**OpenMP**技术，在Master-Worker并行模式下完成文档分类向量化工作。

本实验的主要任务包括：

1. 在虚拟机上搭建、部署MPI、OpenMP环境。
2. 读取指定目录下的若干文档及字典文件。
3. 分别使用MPI和OpenMP进行多趟实验，统计各文档内每个字典单词的出现个数及不同进程数下的运行时间。其中，Master进程负责读取文件目录、分配文档处理任务给Worker进程、接收Worker进程处理结果并进行汇总；Worker进程负责读取及广播字典、从字典文件中构造哈希表、完成Master进程指定的文档处理任务，并向其传送处理结果。
4. 根据上述统计结果得到文档向量，将之存放至文档向量信息结果文件中。
5. 使用小规模的文档及字典文件分析代码逻辑及运行结果的正确性。
6. 根据在不同进程数下使用MPI和OpenMP的各趟运行时间计算加速比，分析并行处理的运行效果。

## 2. 实验目的

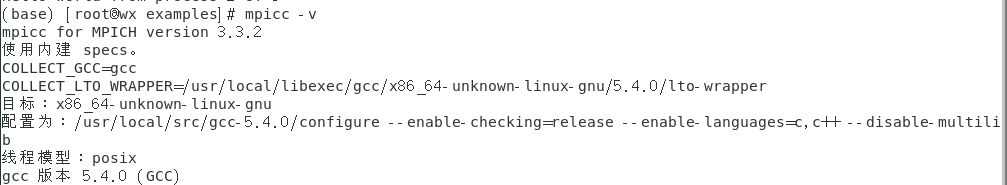
1. **探索较优的并行策略**：MPI和OpenMP技术提供了不同的并行化策略。本实验拟尝试不同的并行化策略，评估它们对文档分类向量化的性能影响，进而了解不同并行化策略在文档分类向量化任务中的优劣势，确定较优的并行策略。
2. **大规模数据处理**：文档分类向量化任务涉及到较大规模的文本数据集。本次实验拟在较大规模的数据处理场景进行分类处理任务，评估MPI和OpenMP技术在处理大规模数据时的可扩展性和效率。
3. **分析不同并行编程模型及进程数完成任务的加速效果**：使用MPI和OpenMP模型完成文档分类向量化过程的并行化，根据各自的运行时间可以比较串行和并行、MPI和OpenMP的性能差异，较为准确地评估加速效果。

## 3. 实验环境

* AMD Ryzen 7 4700U with Radeon Graphics 2.00 GHz + CentOS 7（64位）



* gcc version 5.4.0
* mpicc for MPICH version 3.3.2



## 4. 实验步骤

### 4.1 搭建MPI运行环境

#### 升级gcc版本

系统自带的gcc的版本并不能满足并行编程的需要，我们需要在此基础上升级gcc版本以满足需求。

*#使用清华源到官网下载压缩包*

wget https://mirrors.tuna.tsinghua.edu.cn/gnu/gcc/gcc-5.4.0/gcc-5.4.0.tar.gz

*#将压缩包移动到/usr/local目录下解压*

mv gcc-5.4.0.tar.gz /usr/local

cd /usr/local

tar -zxvf gcc-5.4.0.tar.gz

*#下载编译依赖*

cd gcc-5.4.0

./contrib/download\_prerequisites

*#创建编译目录并生成Makefile*

mkdir gcc-build-5.4.0

cd gcc-build-5.4.0

../gcc-5.4.0/configure --prefix=/usr/local/gcc --enable-checking=release --enable-languages=c,c++ --disable-multilib

*#编译安装*

make -j64 # -j jobs 同时运行该命令的个数 使用多个处理器编译，提高效率

make install

*#修改软链接，将默认gcc修改为安装的指定版本*

rm -rf /usr/bin/gcc

rm -rf /usr/bin/g++

ln -s /usr/local/gcc/bin/gcc /usr/bin/gcc

ln -s /usr/local/gcc/bin/g++ /usr/bin/g++

#### 安装mpicc

mpicc 是MPI的C语言编译器命令。mpicc 命令用于将使用MPI编写的C语言源代码编译为可执行文件。使用 mpicc 命令编译MPI编写的C代码时，常需要链接MPI库以提供并行计算和通信所需的功能。

*#到官网下载压缩包*

wget http://www.mpich.org/static/downloads/3.3.2/mpich-3.3.2.tar.gz

*#解压相应压缩包*

tar -zxvf mpich-3.3.2.tar.gz

*#进入解压后的文件夹内*

cd mpich-3.3.2

*#设置安装的路径*

./configure --prefix**=**/usr/local/mpich-3.3.2

*#进行配置*

make

make install

075be5445e99451f58b4f2f1741d902

#### 配置mpicc相关环境参数

*#修改配置文件*

vim ~/.bashrc

*#添加相关路径*

export PATH**=**"/usr/local/mpich-3.3.2/bin:$PATH"

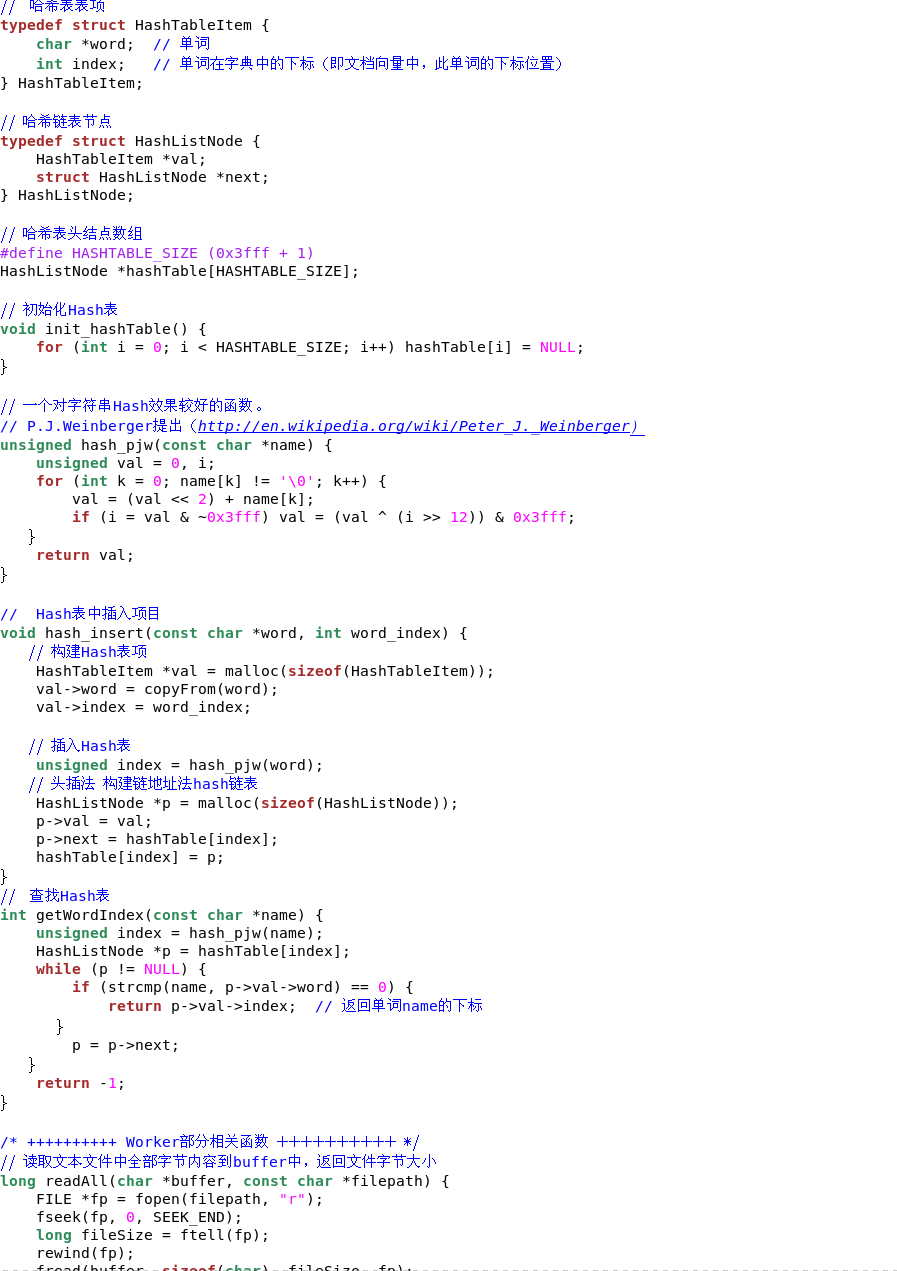
source ~/.bashrc

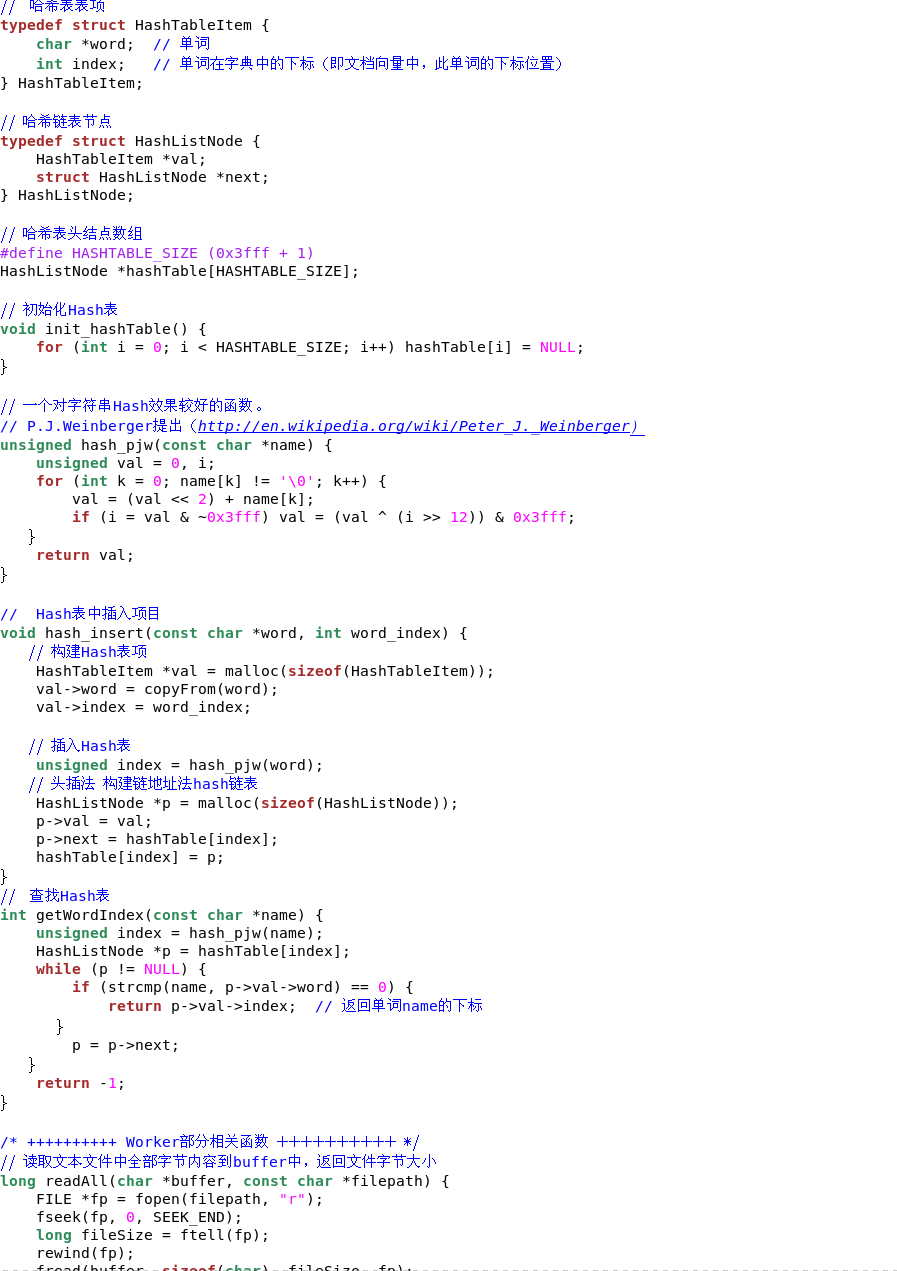
### 4.2 创建源代码

#### MyUtils.c

##### 处理字典和哈希表相关数据结构及函数

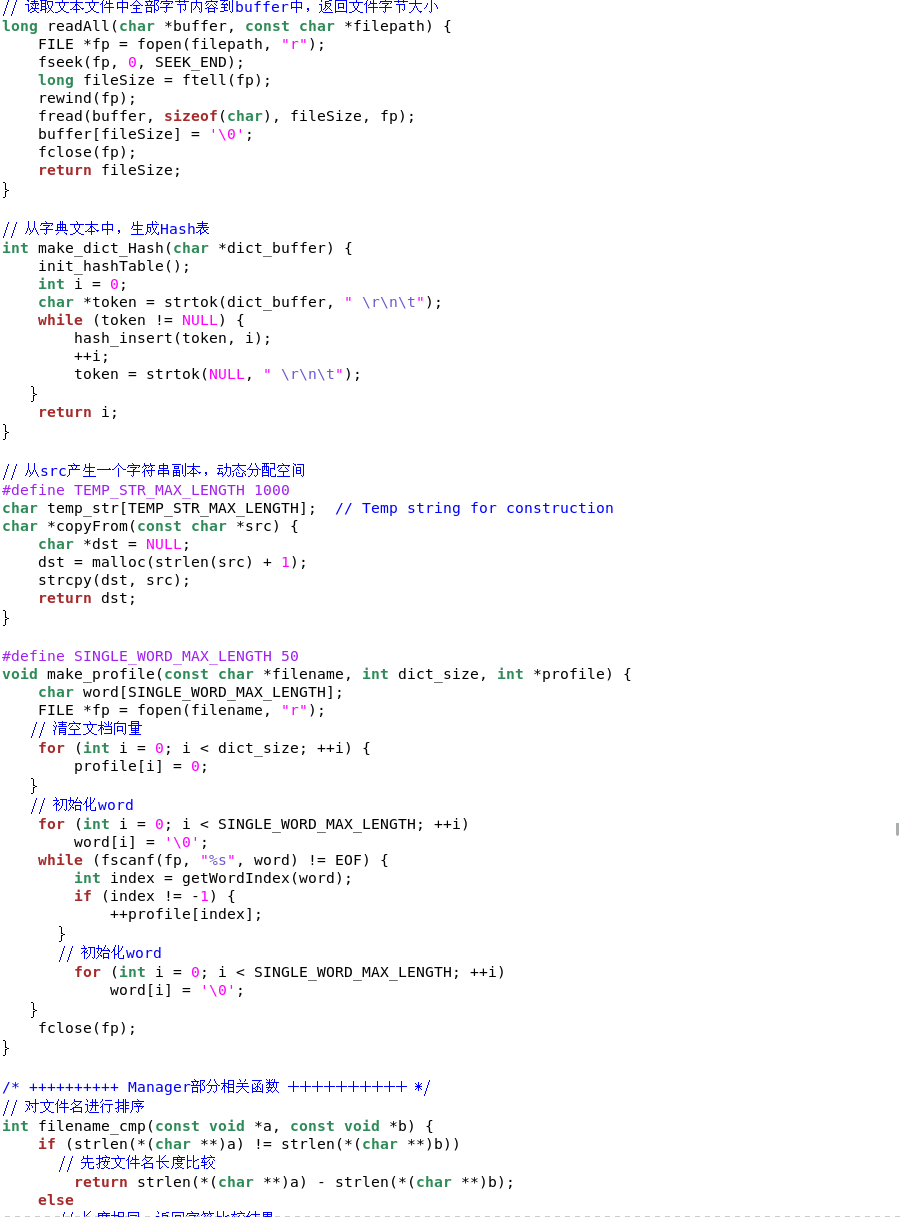
* 定义HashTableItem结构体，表示哈希表的表项。
* 定义HashListNode结构体，表示哈希链表的节点。
* 定义哈希表头结点数组hashTable，指定大小为HASHTABLE\_SIZE。
* 声明init\_hashTable函数，用于初始化hashTable数组的元素为NULL。
* 声明hash\_pjw函数，用于计算字符串的哈希值。
* 声明hash\_insert函数，用于向哈希表中插入项目。
* 声明getWordIndex函数，用于在哈希表中查找单词并返回其下标。





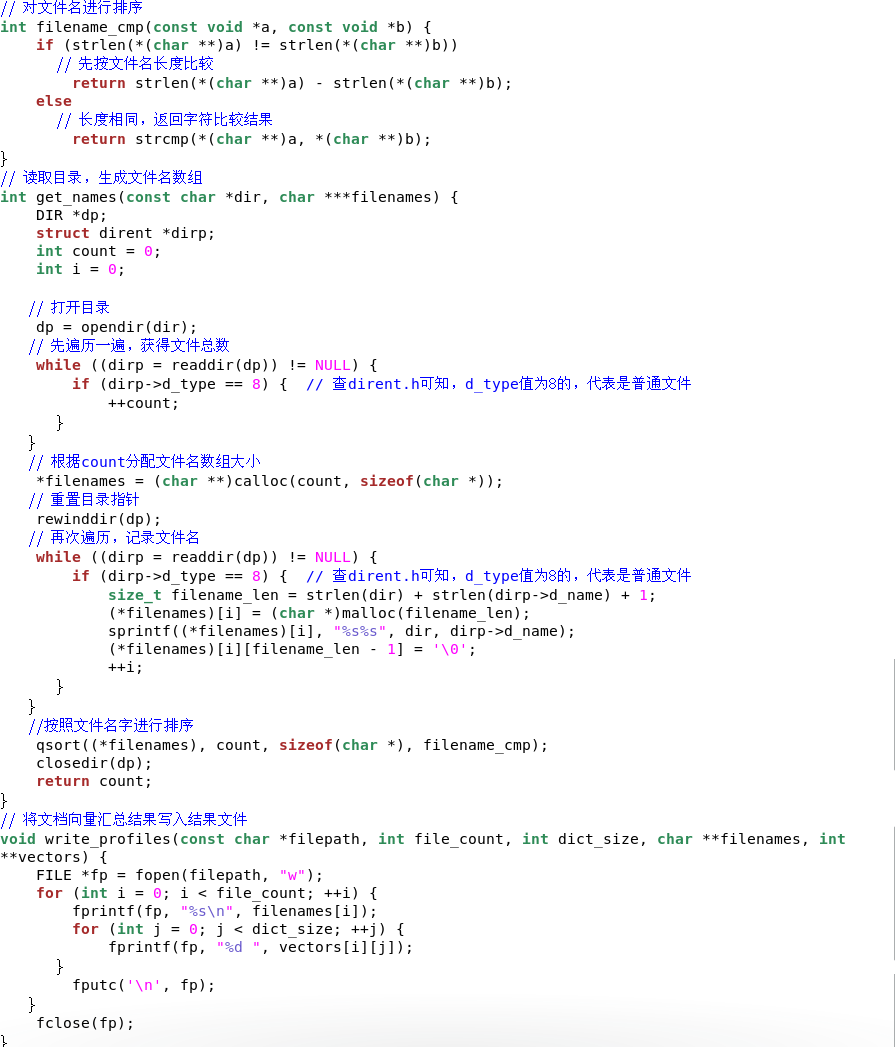
##### Worker相关部分

* 声明readAll 函数，用于读取文本文件到缓冲区，并返回文件字节大小。
* 声明make\_dict\_Hash 函数，用于从字典文本中生成哈希表
* 声明copyFrom 函数，用于从源字符串生成一个动态分配的字符串副本
* 声明make\_profile 函数，用于统计各单词的出现频次，生成文档向量。



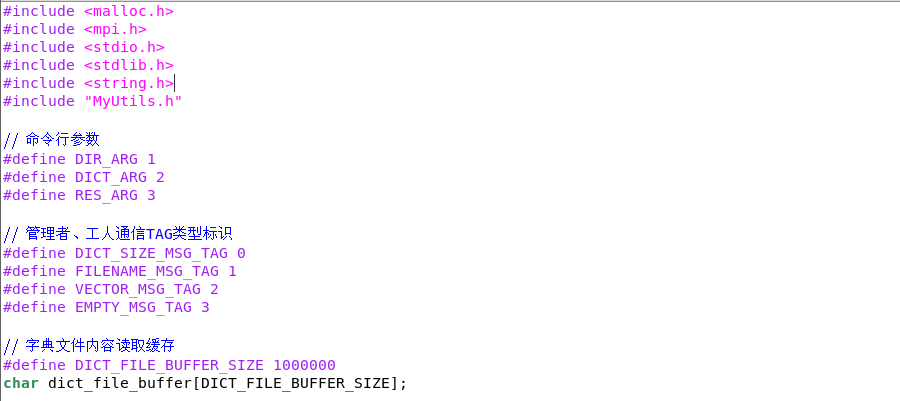
##### Master相关部分

* 声明filename\_cmp 函数，用于对文件名进行比较及排序。
* 声明get\_names 函数，用于读取目录、生成文件名数组，并按文件名进行排序。
* 声明write\_profiles 函数，用于将文档向量汇总结果写入结果文件。



#### MPI的Main.c

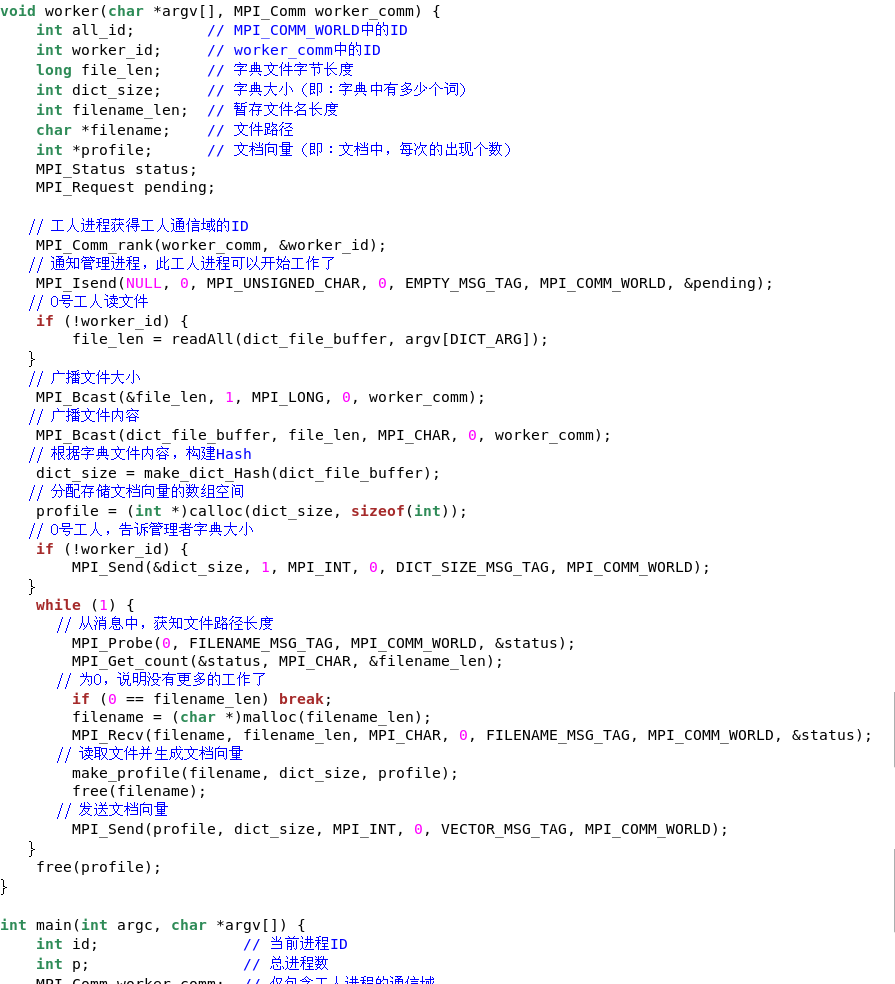
##### 引入库及宏定义



##### Worker函数

Worker函数负责接收任务并处理文档向量化，其主要任务有：

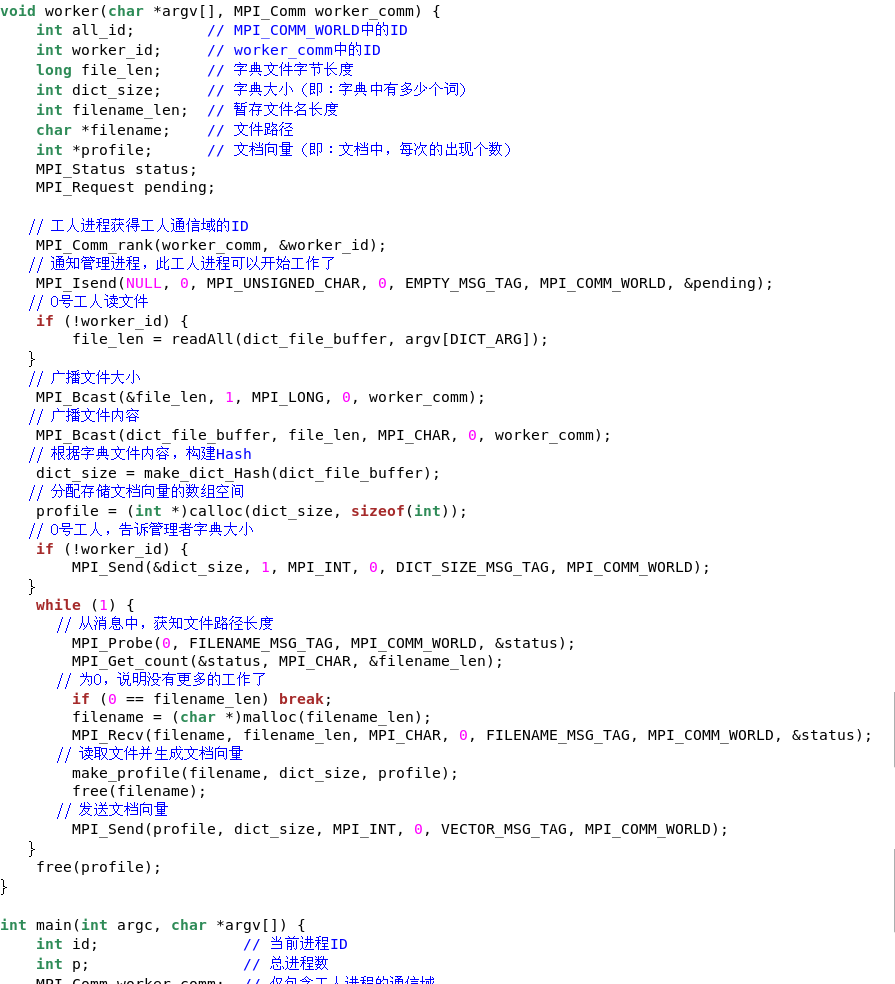
1. 对于普通进程，需要告知0号进程当前Worker进程可以开始工作。
2. 对于0号进程，可以读取、广播字典文件给其他Worker进程。
3. 根据字典文件内容构建哈希表。



##### Manager函数

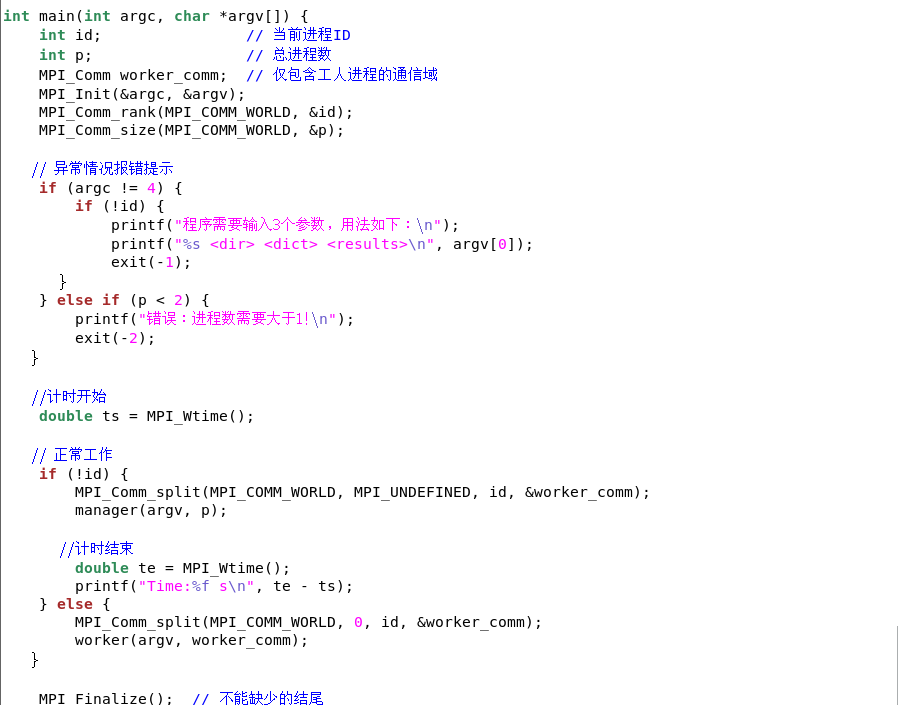
Manager函数负责管理工作进程，进行进程间的通信，其主要任务有：

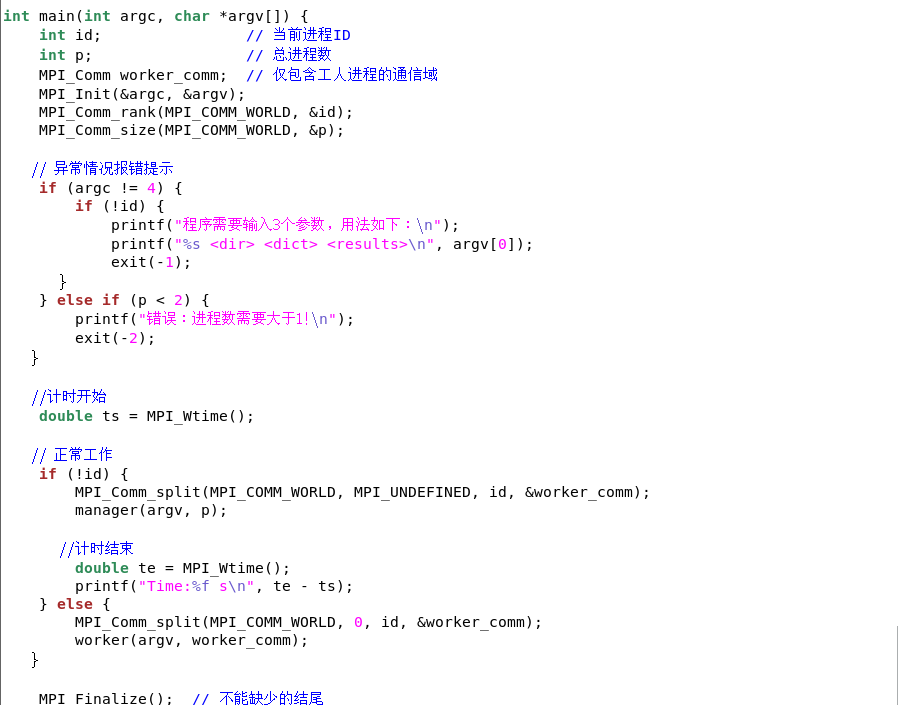
1. 接收Worker进程发送的字典信息。
2. 获取目标目录下的文件名列表。
3. 分配存放文件文档向量的空间。
4. 处理Worker进程发送的消息并将结果写入文件。



##### Main函数

Main函数是程序的入口点，负责进行MPI的初始化，并根据进程ID执行不同的逻辑：如果是0号进程，调用Manager函数进行Master进程逻辑；如果是其他进程，调用Worker函数进行Worker进程逻辑。



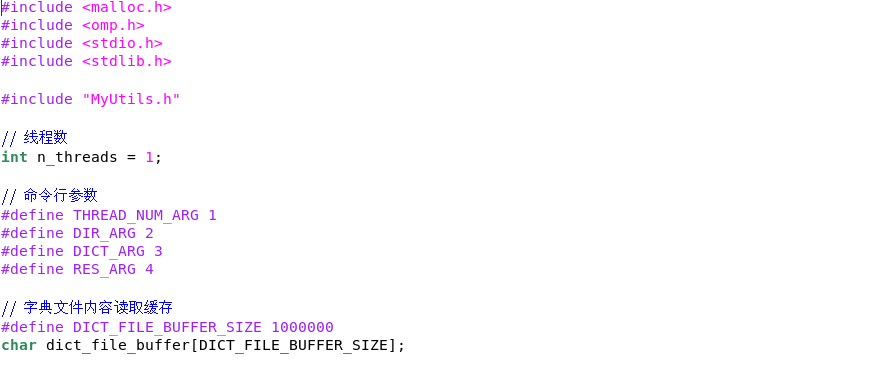


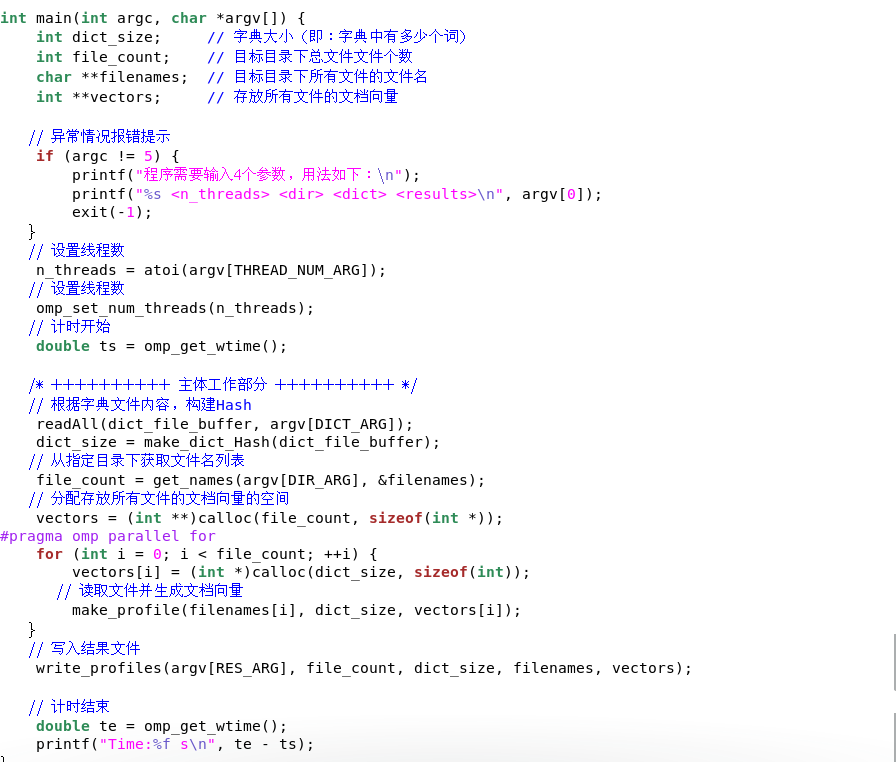
#### (3) OpenMP的Main.c

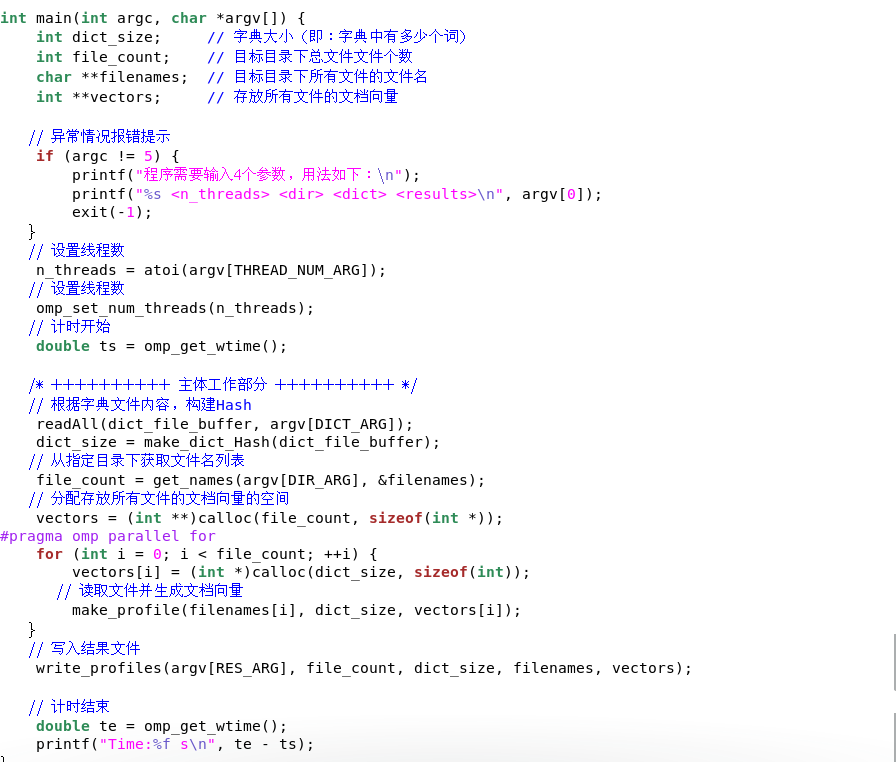
##### Main函数

Main函数是程序的入口点，接收命令行参数及进行初始化，主要负责：

1. 检查命令行参数是否满足要求，若不满足，则输出提示并退出程序。
2. 从命令行参数中获取线程数（n\_threads）。
3. 设置OpenMP的线程数为n\_threads。
4. 记录程序执行时间。
5. 读取字典文件内容并由此构建哈希表，从指定目录下获取文件名列表，进行文件处理，最终分配存储空间，生成文档向量，并将结果写入文件。



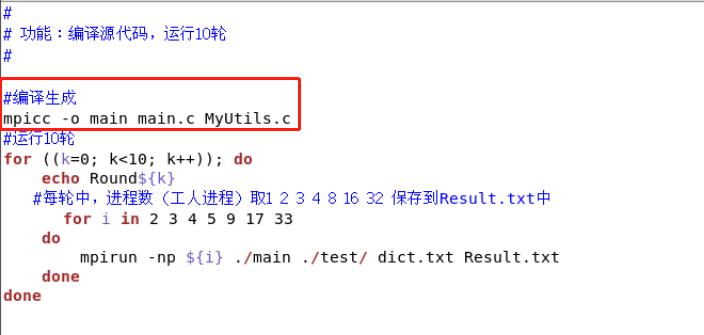




### 4.3 进行编译

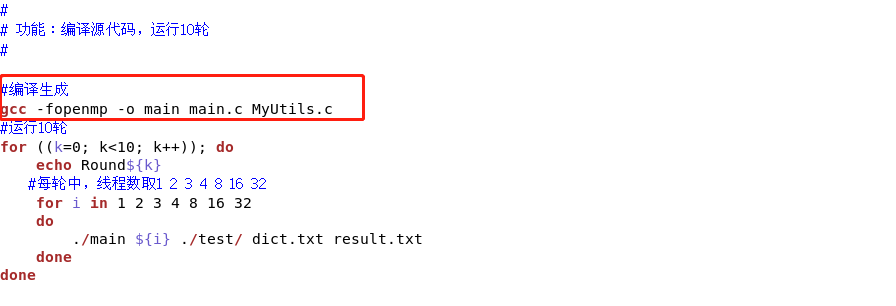
#### MPI

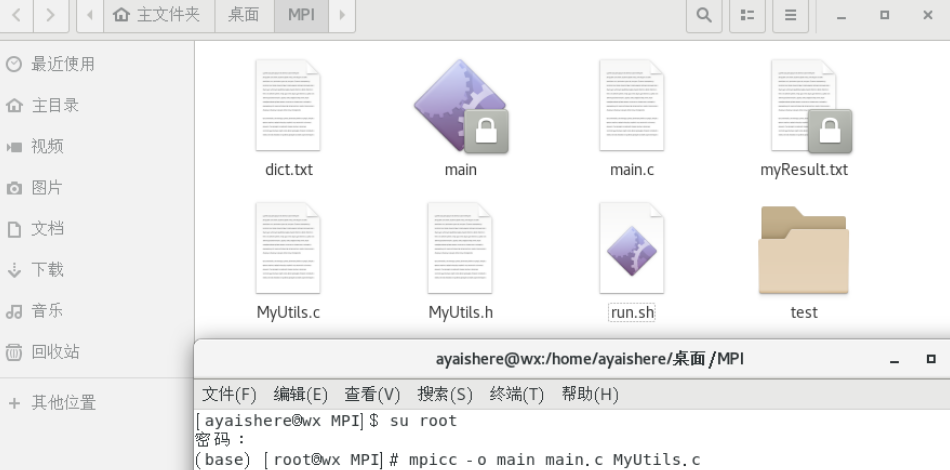
使用shell脚本run.sh进行编译。

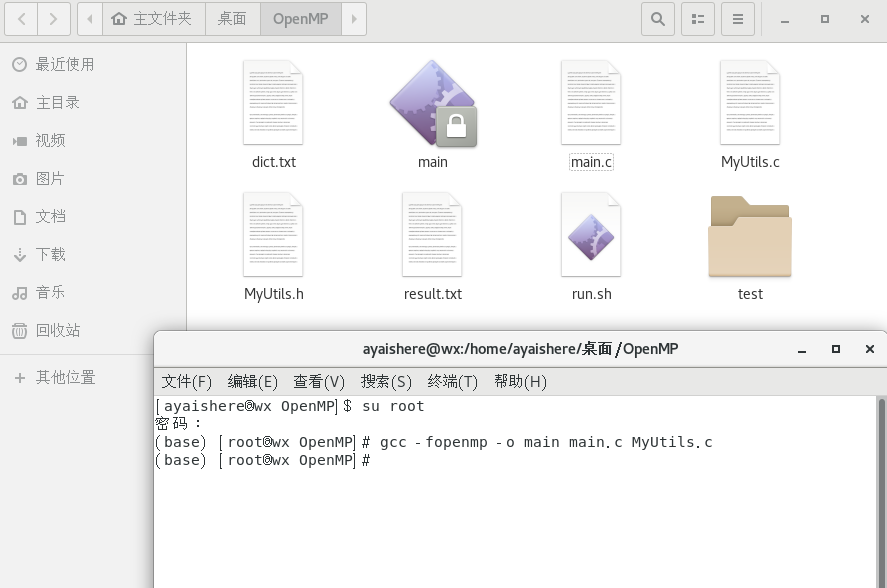


#### OpenMP

使用shell脚本run.sh进行编译。







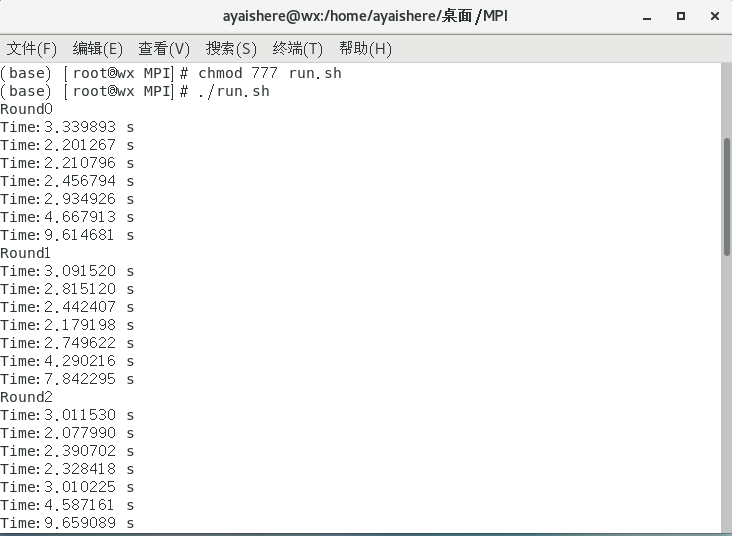
### 4.4 运行程序

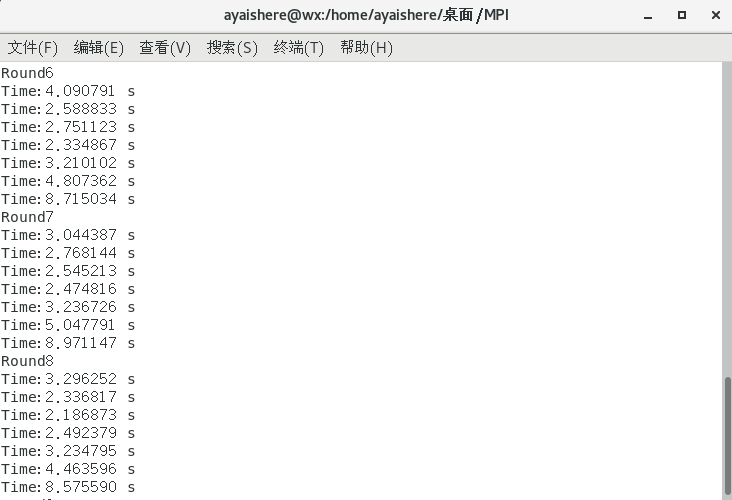
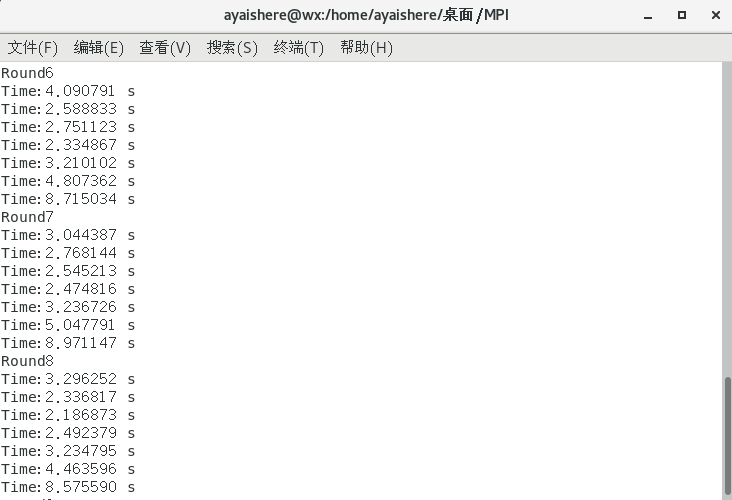
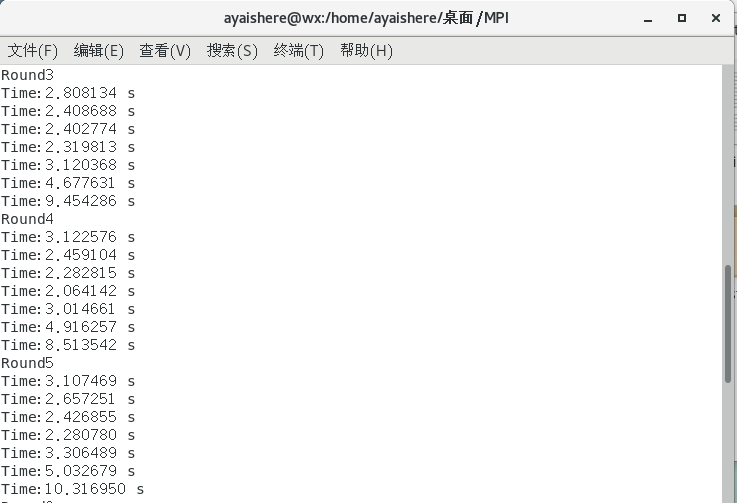
为了减少误差，我们运行脚本以运行目标并行程序，完成文档向量化工作。实验中，分MPI和OpenMP两个模型分别进行实验。

#### MPI

使用shell脚本run.sh运行程序。为减少误差，采用Master-Worker模式，使用不同的Worker进程个数进行实验，计算加速比。每一轮中，Worker进程数分别取 1、2、3、4、8、16 和 32，共进行10趟实验。

每一趟中各进程数对应的运行时间如下：

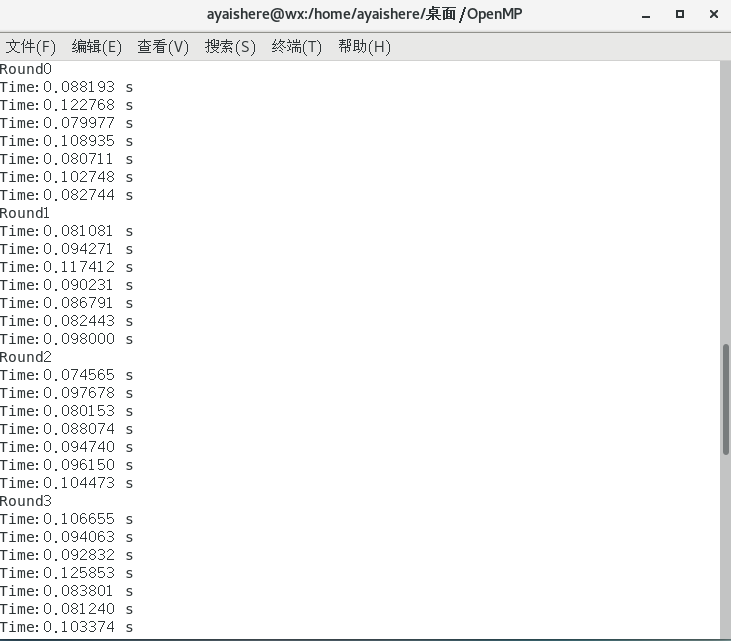


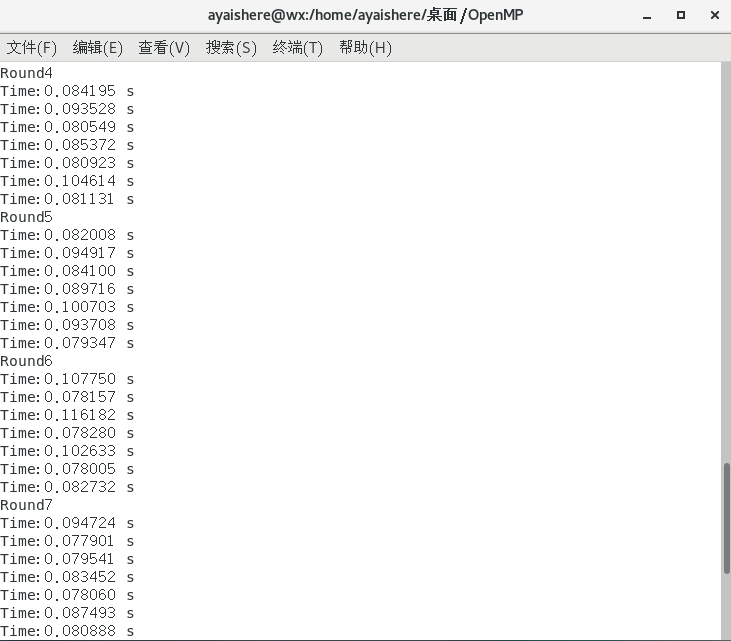


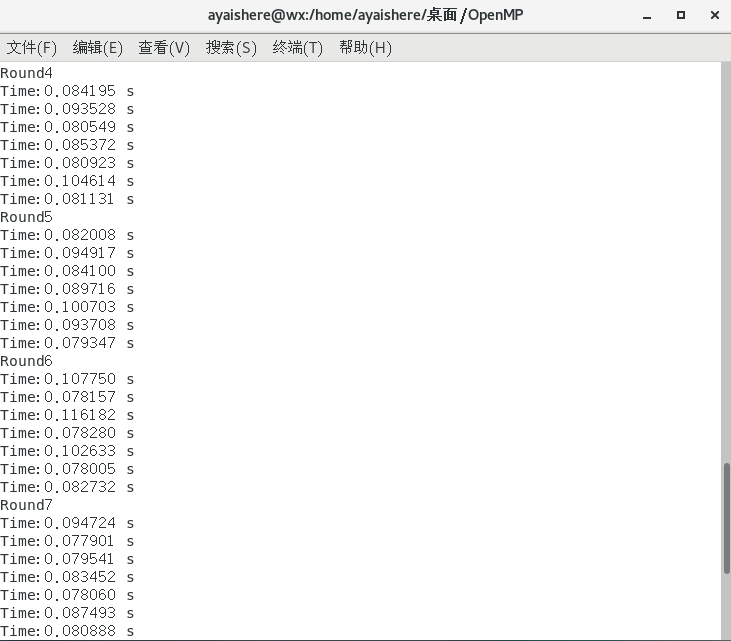


#### OpenMP

使用shell脚本run.sh运行程序。为减少误差，采用Master-Worker模式，使用不同的Worker进程个数进行实验，计算加速比。每一轮中，（工人）进程数分别取 1、2、3、4、8、16 和 32，共进行10趟实验。





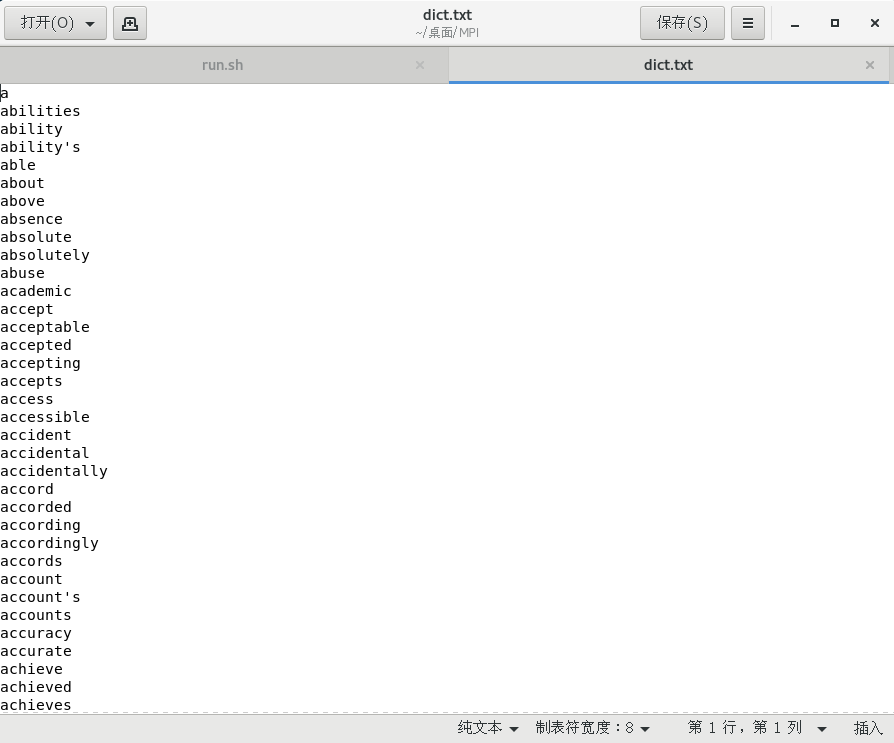




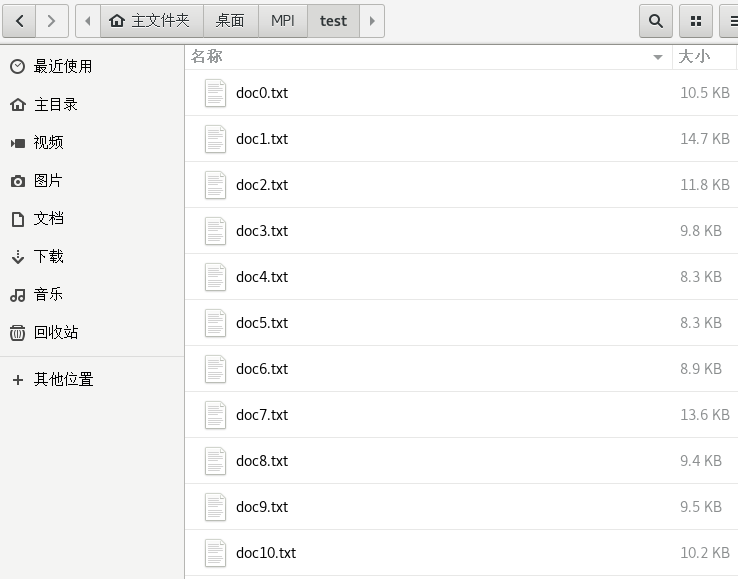
## 实验分析

### 5.1 实验内容分析

本实验中的字典dict.txt格式如下，共4374个单词，按照首字母从‘a’到‘z’进行排序：

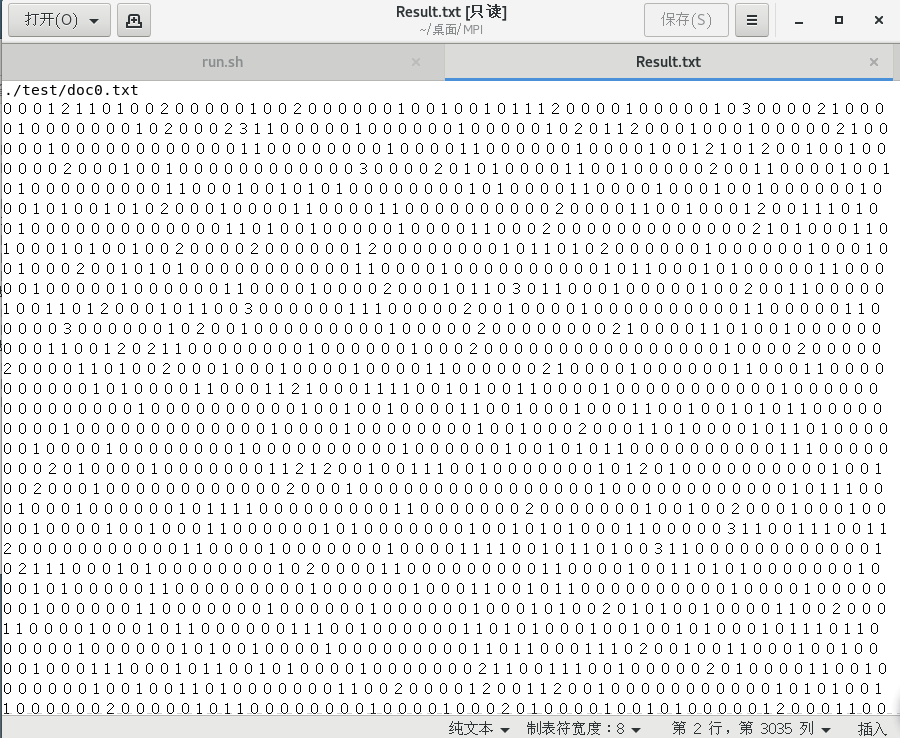


本实验中的文档docx.txt格式如下（以doc0.txt为例），共100个文档，由doc0.txt命名到doc99.txt：





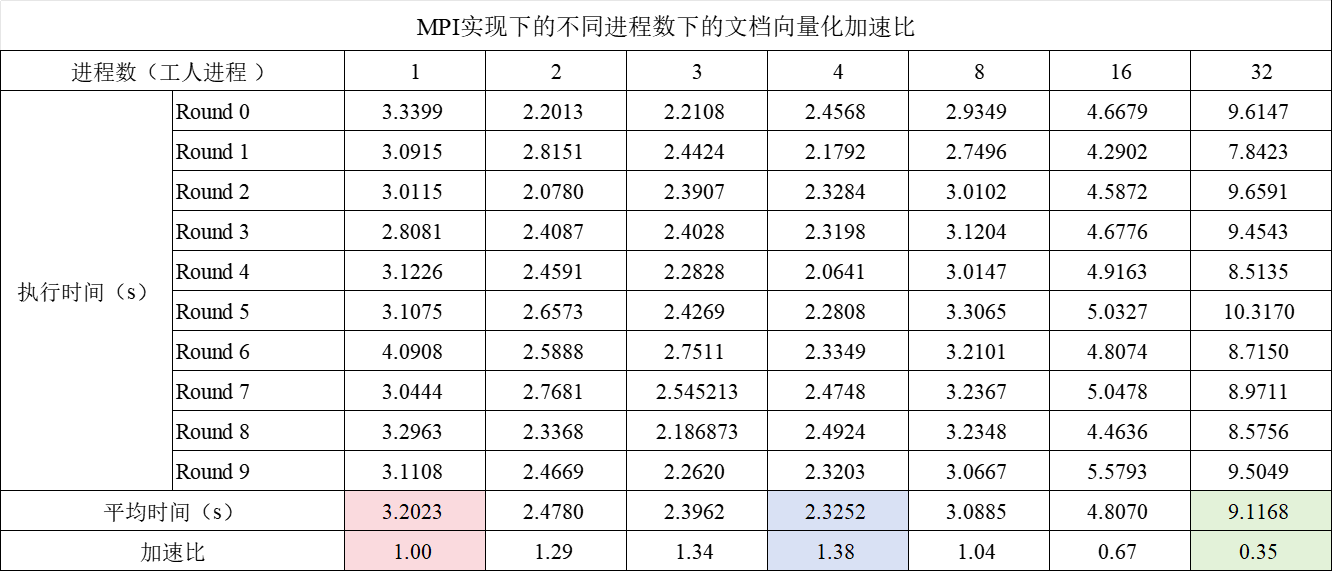
最终的文档分类运行结果为（以doc0.txt为例）：



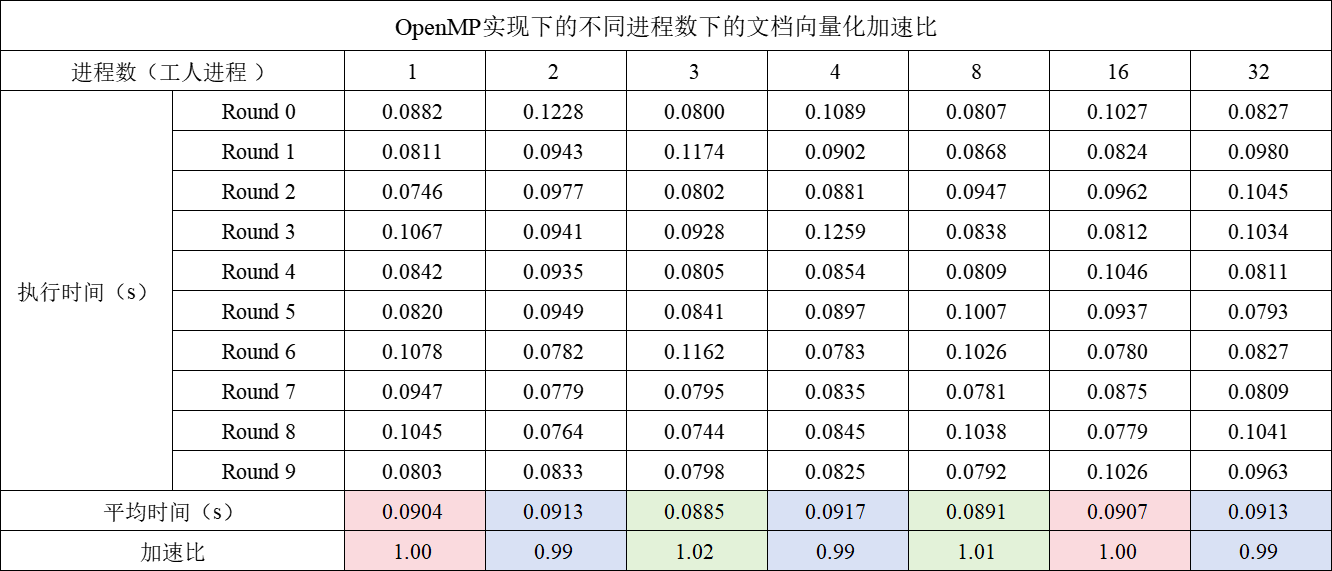
### 5.2 实验结果分析

MPI和OpenMP两种不同模型的运行时间及加速比如下：

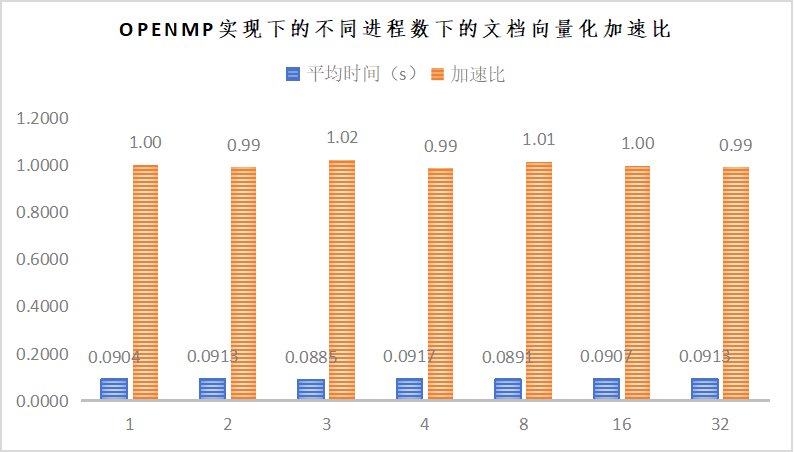
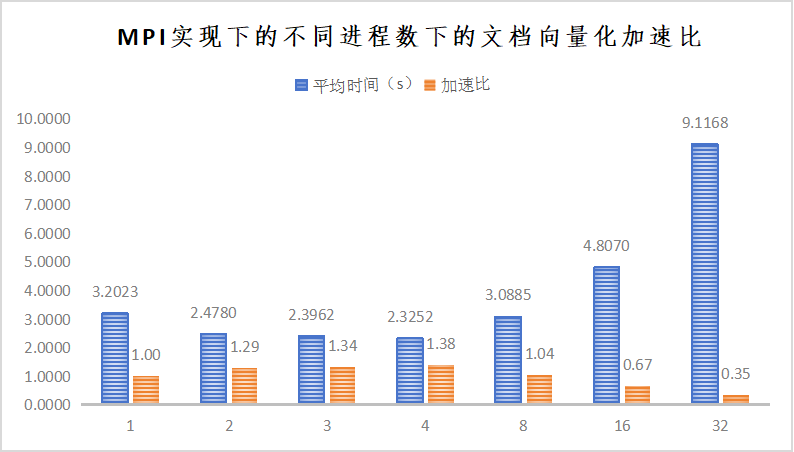
* MPI：



* OpenMP：



根据上表绘制柱状图，直观分析两种模式下的并行处理效率：



本次实验中，由于采用的哈希方法较为高效，计算时间均不是很大，各趟单个运行过程的时间基本控制在10s内。

总体来看，MPI和OpenMP模式下的加速比均不是很理想，并行的额外开销比较大。这一点在MPI实现中有很大的体现。

单独来看，MPI实现的实验结果符合预期，即随着进程数增加，加速比逐渐增大到最大值，再随着进程数的增大，加速比下降。这是由于，随着进程数的增加，模型能充分利用2个物理CPU核的并行资源，加速比也就接近于最高。而进程数增加到一定程度，由于物理CPU核有限，再增加线程数也无法从物理上增加程序运行的并行程度，反而因为进程数的增加，带来进程通信和管理等的额外开销，所以使得运行时间增加，加速比下降。同时，后期下降的幅度很大，显示出增加进程带来的额外开销增长较为明显。在MPI程序中，进程通信的开销主要体现在工人进程之间广播字典文件内容，其开销与进程个数是线性关系，当进程大到一定程度后，会成为主要开销。随着进程数从4、8、16不断增大到32时，加速比每次刚好差不多是前一次的一半，正好可以验证想法的正确性。

而OpenMP是利用多线程来加速程序执行，在共享内存系统上运行。相较于进程，线程是更轻量级的，可以提供更细粒度的并行，且通信等的额外开销更小。由于本次实验中OpenMP程序没有涉及线程间数据通信，结果中OpenMP显示出的并行额外开销小，即随着线程增大，加速比没有明显下降。

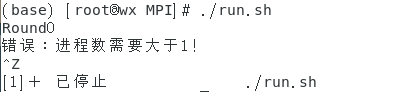
## 问题与讨论

1. 安装配置mpich过程中报错： make: \*\*\* 没有指明目标并且找不到 makefile

* 问题分析及解决方案：mpicc包中有C语言和Fortran语言的实例，但我的虚拟机上没有装Fortran的编译器，因为无法通过make。可以通过禁用fortran实现处理。

./configure --disable-fortran

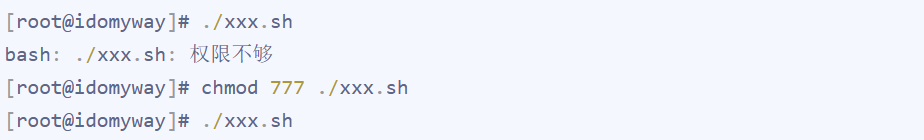
1. 运行过程中进程数设置不合理



* 问题分析及解决方案：进程数设置不合理，增大进程数即可运行。

1. 切换到root状态执行脚本仍然显示权限不够

* 问题分析及解决方案：chmod 777 ./run.sh以给予脚本程序足够的权限。



## 实验心得

通过文档分类向量化的实验，我们更加深入地学习了文本分类的相关知识及相应的分类方法。通过实验，我们获得了宝贵的编程实践经验，对并行编程原理也有了更深入的理解，团队整体的编程能力也得到了提高。总之，这是一次让我们收获颇丰的实践学习经验。

## 参考资料

1. <http://www.tup.com.cn/bookscenter/book_01238001.html>
2. <https://blog.csdn.net/weixin_42427540/article/details/117114611>
3. <https://zhuanlan.zhihu.com/p/356705583>
4. <https://zhuanlan.zhihu.com/p/355850573#SnippetTab>
5. <https://blog.csdn.net/jiacong_wang/article/details/105593209>
6. <https://blog.csdn.net/susandama73/article/details/119651107>
7. <https://blog.csdn.net/qq_36819130/article/details/88370973>
8. <https://blog.csdn.net/linpengzt/article/details/115478648>
9. <https://blog.csdn.net/b_ingram/article/details/121569398>
10. <https://blog.csdn.net/lusongno1/article/details/61709460>
11. <https://blog.csdn.net/idomyway/article/details/108229822>