《时空数据处理与组织》

课程设计报告

学　　院：　遥感信息工程学院

班　　级：　20F10

学　　号：　2020302131249

姓　　名：　马文卓

2022 年 5 月 12 日

目录

一、软件安装及数据导入

----------任务及要求----------

任务：在同一台机器上，分别安装 postgis 和 mongodb 服务器程序，并能正确启动，然后导入 gdam 数据（20 分）

要求：

①说明安装过程，并创建 test 数据库，用自己名字的全拼作为导入数据的表名（postgis）和集合名（mongodb）（5 分）；

②导入方法不限，但需要说明从原始数据导入到两个数据库的整个过程（10 分）；

③postgis 导入过程可直接建立空间索引，mongodb 先不要建立索引；

④报告中应在本部分反映出实验的软硬件环境，例如设备的 CPU，内存，操作系统版本等（5 分）。

----------实施步骤----------

1. 软硬件环境介绍

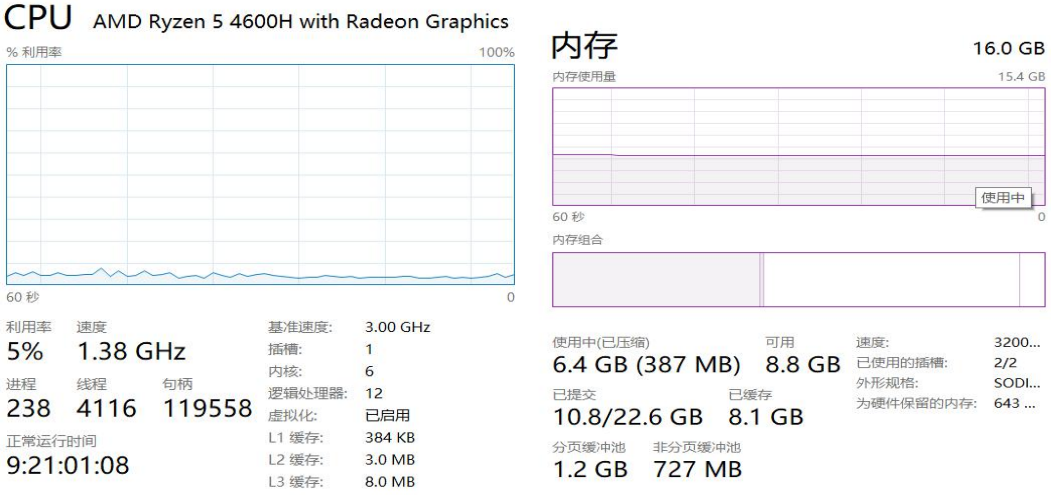
电脑型号：Lenovo Legion R7000 2020

CPU:AMD Ryzen 5 4600H with Radeon Graphics（6 内核）

运行内存：16GB　　储存内存：100GB（C 盘）+375GB（D 盘）

显卡：NVIDIA GeForce GTX 1650 Ti

操作系统版本：64 位 win10 系统

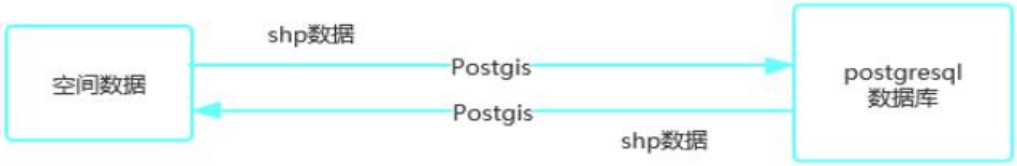


本次项目中所有程序均为【python】语言编写，版本为 3.9.7，具体依赖包在程序讲解前均有叙述。

2. postgis、mongdb 安装

（1）postgis 安装

因为空数据具有空间位置、空间关系、分类编码、海量数据的特征，所以一般的数据库管理系统无法存储空间数据，PostgreSQL 也是如此。为了能够存储空间数据，采用“关系型数据库+空间数据引擎”的解决方案。如此说来，PostGis 和 PostgreSQL 的关系可以囊括为：PostGis 在其中为空间数据引擎提供支持。它为 Postgresql 提供对空间数据类型、空间索引和空间函数等的空间信息服务，将 PostgreSQL 转变为可以储存空间数据的数据库。因此在安装 PostGis 之前先安装 PostgreSQL。



①安装 postgreSQL

1>进入 postgreSQL 下载官网，选择对应版本（version-14.2windowsX86-64）

https://www.postgresql.org/download/windows/

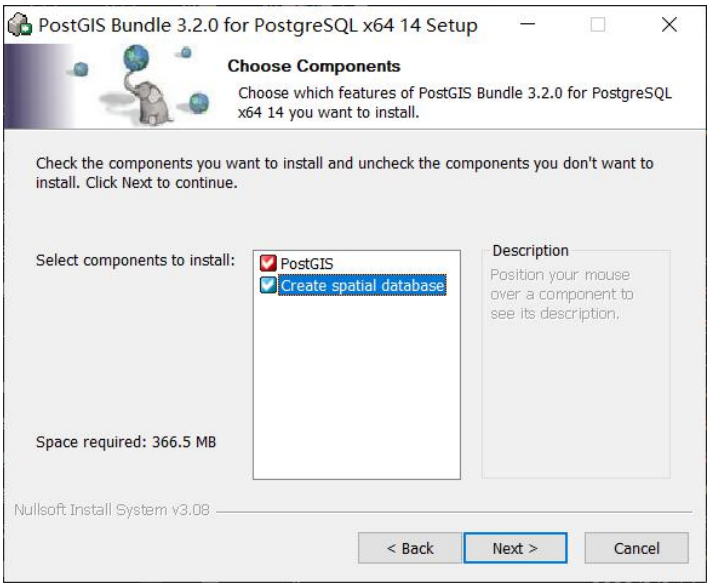
2>选择好下载目录，输入密码，设置端口号（5432），安装完成。

②安装 postGis

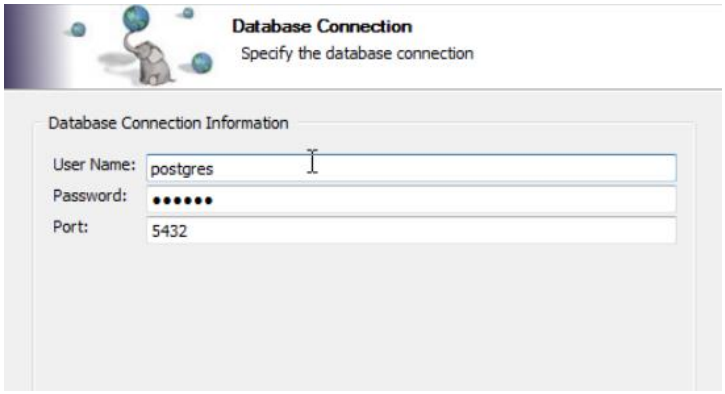
1>进入官网下载地址，选择 PostGis3.2 进行下载

http://download.osgeo.org/postgis/windows/pg10/

2>选择 Create spatial database，初始化一个空间数据库



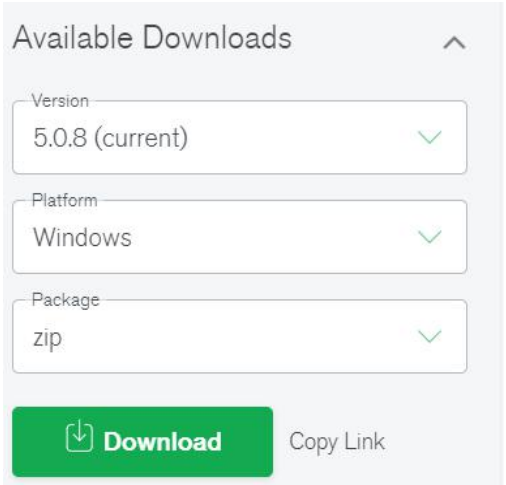
3>输入用户名（postgres）、密码、端口号（5432）



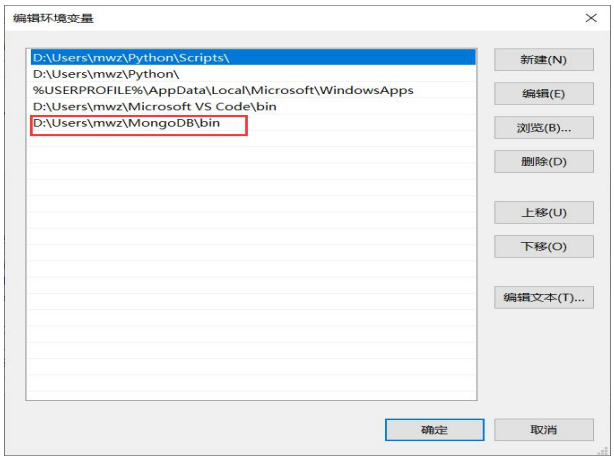
4>初始化空间数据库名称后安装完毕

（2）mongdb 安装

1>进入官网（www.mongodb.com）选择合适版本进行下载



2>等待安装完毕，在电脑=>属性=>高级系统设置=>环境变量=>Path 中添加 MongDB 的 bin 路径（这样无论在什么路径下均可进入 MongDB 啦）



3>查看是否安装成功：进入 cmd 命令行界面，输入 mongo 进入数据库。

```
C:\Users\mwz>mongo
MongoDB shell version v5.0.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("2787e195-7fa3-4602-9088-29c2c1e8b050") }
MongoDB server version: 5.0.6
================
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
================
---
The server generated these startup warnings when booting:
        2022-04-27T18:55:26.346+08:00: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
---
---

        Enable MongoDB's free cloud-based monitoring service, which will then receive and display
        metrics about your deployment (disk utilization, CPU, operation statistics, etc).

        The monitoring data will be available on a MongoDB website with a unique URL accessible to you
        and anyone you share the URL with. MongoDB may use this information to make product
        improvements and to suggest MongoDB products and deployment options to you.

        To enable free monitoring, run the following command: db.enableFreeMonitoring()
        To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

3. 数据导入

（1）PostGis 导入数据

①在 PostgreSQL 的 bin 目录下使用【createdb -U postgres -E UTF8 -e test】命令创建 test 数据库。然后使用【psql -U postgres -d test】命令连接到数据库，最后使用【CREATE EXTENSION postgis;】和【CREATE EXTENSION postgis_raster;】创建扩展。

```
D:\Users\mwz\PostgreSQL\bin>createdb -U postgres -E UTF8 -e test
口令:
SELECT pg_catalog.set_config('search_path', '', false);
CREATE DATABASE test ENCODING 'UTF8';
```

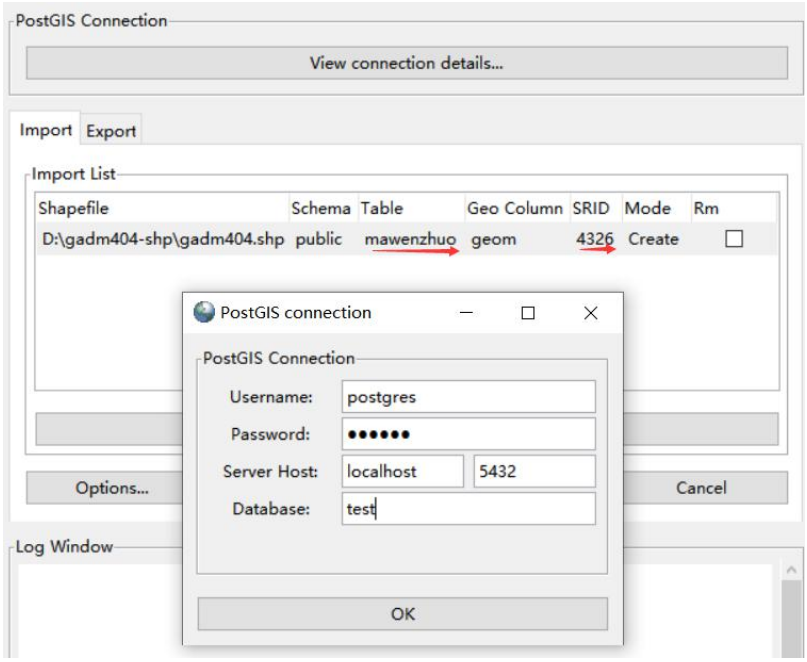
```
D:\Users\mwz\PostgreSQL\bin>psql -U postgres -d test
用户 postgres 的口令:
psql (14.2)
输入 "help" 来获取帮助信息.

test=# CREATE EXTENSION postgis;
CREATE EXTENSION
test=# CREATE EXTENSION postgis_raster
test-# CREATE EXTENSION postgis_raster;
错误:  语法错误 在 "CREATE" 或附近的
第2行CREATE EXTENSION postgis_raster;
       ^
test=# CREATE EXTENSION postgis_raster;
CREATE EXTENSION
test=# \l
                                                数据库列表
      名称       | 拥有者 | 字元编码 |          校对规则          |          Ctype             |       存取权限
-----------------+--------+----------+----------------------------+----------------------------+-----------------------
 mawenzhuo       | postgres | UTF8   | Chinese (Simplified)_China.936 | Chinese (Simplified)_China.936 |
 postgis_32_sample | postgres | UTF8 | Chinese (Simplified)_China.936 | Chinese (Simplified)_China.936 |
 postgres        | postgres | UTF8   | Chinese (Simplified)_China.936 | Chinese (Simplified)_China.936 |
 template0       | postgres | UTF8   | Chinese (Simplified)_China.936 | Chinese (Simplified)_China.936 | =c/postgres          +
                                                                                                  | postgres=CTc/postgres
 template1       | postgres | UTF8   | Chinese (Simplified)_China.936 | Chinese (Simplified)_China.936 | =c/postgres          +
                                                                                                  | postgres=CTc/postgres
 test            | postgres | UTF8   | Chinese (Simplified)_China.936 | Chinese (Simplified)_China.936 |
(6 行记录)
```

②使用 ogr2ogr 命令导入 gadm404.shp，但是出现了如下错误。

```
D:\Users\mwz\QGIS\bin>ogr2ogr -f PostgreSQL "PG:dbname=test user=postgres password=090012" -lco PG_USE_COPY=YES -lco SHAPE_ENCODING=GBK -progress -update -append  -gt -1 -nln ogr_china  D:/本科/时空数据处理与组织/期末设计/gadm404-shp/gadm404.shp
ERROR 1: PROJ: proj_identify: D:\Users\mwz\PostgreSQL\share\contrib\postgis-3.2\proj\proj.db contains DATABASE.LAYOUT.VERSION.MINOR = 0 whereas a number >= 2 is expected. It comes from another PROJ installation.
```

于是改用 postGis 可视化导入工具，导入 gadm404.shp，表名为 mawenzhuo，坐标系设置为 WGS84。

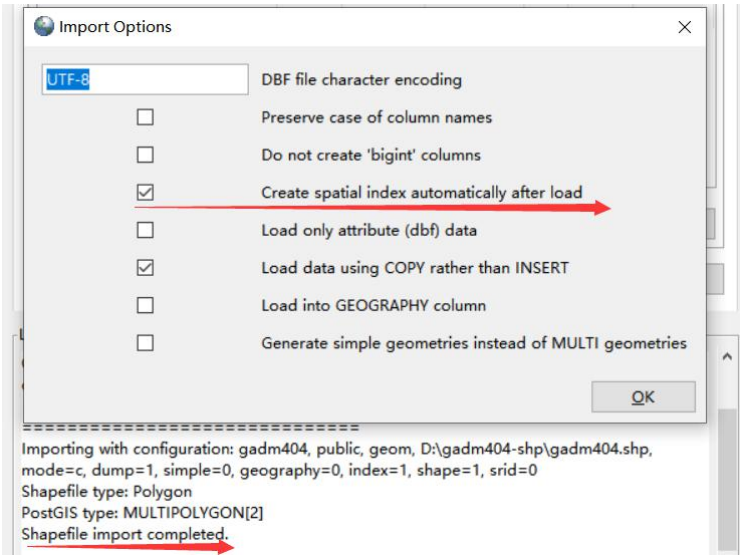


但是出现了无法打开 dbf 文件的错误

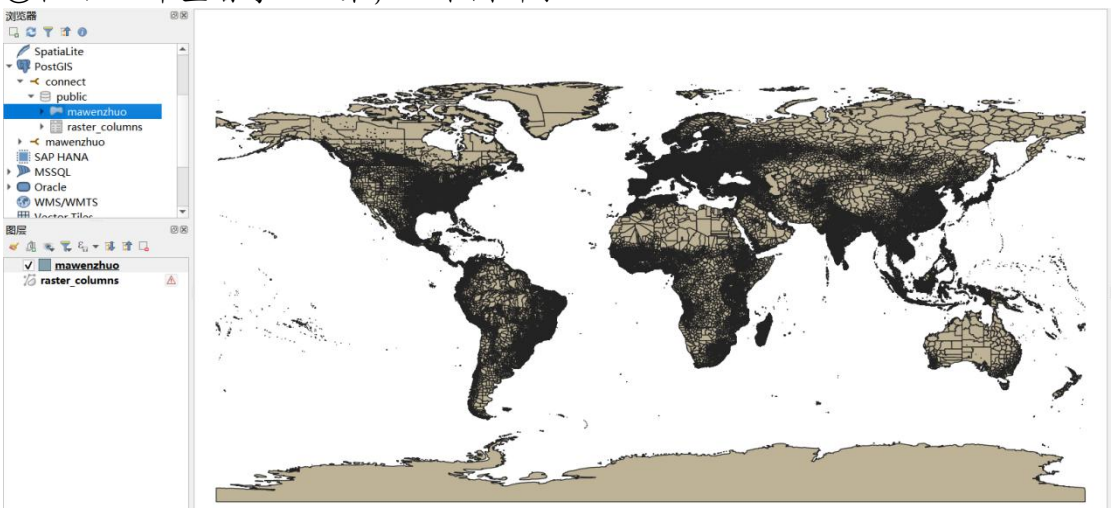
```
Importing with configuration: mawenzhuo, public, geom, D:\本科\时空数据处理与组织\期末设计\gadm404-shp\gadm404.shp,
mode=c, dump=1, simple=0, geography=0, index=1, shape=0, srid=0
D:\本科\时空数据处理与组织\期末设计\gadm404-shp\gadm404.shp: dbf file (.dbf) can not be opened.
Shapefile import failed.
```

经过搜索学习，了解了出现这种错误的三种情况（如下），于是我将 shapefile 文件均放到 D 盘根目录下（且无中文路径），就解决了问题，导入成功（并创建空间索引）。

1. 导入的shp文件路径太深，换成短路径；
2. 导入的shp文件名称/路径中有中文，去掉；
3. 将需要导入数据的shp 文件、dbf 文件、prj 等文件放在到同一个文件夹内，且名字要一致；



③在 QGis 中查看导入结果，如下图所示



（2）MongDB 导入数据

①使用【use test】创建 test 数据库，使用【db.createCollection（‘mawenzhuo’）】新建 mawenzhuo 集合

```
> use test
switched to db test
> db.createCollection('mawenzhuo')
{ "ok" : 1 }
```

②使用【ogr2ogr -f geoJSON】命令，将 gadm404.shp 文件转为 gadm404.json

```
D:\Users\mwz\QGIS\bin>ogr2ogr -f geoJSON D:/gadm404-shp/gadm404.json D:/gadm404-shp/gadm404.shp
```

③由于 MongDB 有单条数据不能超过 16MB 的限制，因此采用 python 语言编写 load_data 函数【逐行读入】数据。但是遍历尝试发现 json 数据中有三行数据仍然超过 16MB，于是采用跳过这三条数据的方式进行读取。具体程序编写步骤如下（完整程序见附录 load_data.py，此程序使用了 pymongo、json 第三方库）：

1>打开文件

```
#打开文件
filename="D:\\gadm404-shp\\gadm404.json"
file=open(filename,"r",encoding="utf-8")#以只读的模式打开json文件
```

2>连接数据库

```
#连接mongdb,创建client
client=MongoClient('localhost',27017)
#连接数据库test
db=client.test
#指定集合mawenzhuo
collection=db.mawenzhuo
```

3>逐行读入文件。这里根据文件格式，跳过前面 4 行文件头以及三行超限的数据（经过之前的试验，我得知第 2861、36798、40635 行数据为朝鲜数据）。然后如果每一行倒数第二位为‘，’则为中间行，直接插入数据；如果没有‘，’，但是长度大于 4，则为最后一行；否则读取完毕。

```
#逐行导入文件
index=0#索引
start=5#跳过前面的文件头
for everyline in file:
    index=index+1
    if index<start:#跳过文件头
        continue
    elif index==2861 or index==36798 or index==40635:#跳过超过16M的数据
        continue
    else:
        if everyline[-2]==',':#中间行
            print(index)
            each_data=json.loads(everyline[:-2])#载入每一条数据
            #print(each_data)
            collection.insert(each_data)#插入数据库集合
            print("------------------------------------------------------------------------")
        elif len(everyline)>4:#最后一行
            print('last line:',index)
            each_data=json.loads(everyline[:-1])
            #print(each_data)
            collection.insert(each_data)
            print("------------------------------------------------------------------------")
        else:#读取完毕
            break
```

4>在 mongoDB 中查看数据，一共导入 348901 条数据，大小为 2.08GB，格式如下。

二、性能及索引功能测试

----------任务及要求----------

任务：性能及索引功能测试（20 分）

要求：

①分别在 postgis 及 mongodb 上，检索西经 60 度的经线穿过的所有国家。注意，数据是精确到每个国家的县级数据的，因此对查询结果需要进行综合过滤，只给出国家的名字，并统计检索的时间。多做几次检索（5 次以上），时间取平均值，然后比较 postgis（带空间索引）和 mongodb（无空间索引）的检索效率（10 分）；

②将 mongodb 中的数据建立空间索引，然后重复①中的检索，并统计检索的平均时间，与①中的时间做比较（5 分）；

③所有时间的统计信息最终放在一张表格当中进行统计，统一做性能比较分析（5 分）；

④所有操作可命令行执行，也可用程序实现，但都要给出截图并详细说明

----------实施步骤----------

1. PostGis 检索

任务为检索西经 60 度经线穿越的所有国家，我通过编写 python 程序调用 postgresql 的第三方库 psycopg2 来解决。具体步骤如下（完整程序详见附录 select_postgis.py，使用第三方库为 psycopg2 和 time）：

1>连接数据库函数：这里将其封装为 connect_db 函数，方便使用。使用【psycopg2.connect】方法，与 test 数据库建立连接，成功和返回连接。

```
#连接数据库
def connect_db():
    try:
        conn=psycopg2.connect(database='test',user='postgres',
        password='090012',host='127.0.0.1',port=5432)
    except Exception as e:
        print(e)
    else:
        return conn
    return None
```

2>关闭数据函数：封装为 close_db 函数。

```
#关闭数据库
def close_db(cur,conn):
    conn.commit()#事物提交
    cur.close()
    conn.close()#关闭
```

3>查询数据库：封装为 select_db 函数。首先调用 connect_db 函数连接数据库，然后创建游标对象，使用游标对象执行 SQL 查询。【这里的 SQL 语句指定从 mawenzhuo 表格对 NAME_0（国家名）进行查询，由于要求输出为穿过的国家，因此对国家进行 group by 分组。其中的使用 ST_Intersects 对数据中心所有多边形和西经 60 度（即 LINESTRING（-60 -89.99，-60 89.99））求交集，最后得到的结果即为答案】。然后进行结果输出和时间计算，最后调用 close_db 函数关闭数据库。

```
#查询西经60度的经线穿过的所有国家
def select_db():
    conn=connect_db()#连接数据库test
    if not conn:
        return None
    cur=conn.cursor()#创建cursor对象执行SQL命令
    sql='SELECT NAME_0 from mawenzhuo where ST_Intersects(mawenzhuo.geom,\
                    ST_GeomFromText(\'LINESTRING(-60 -89.99,-60 89.99)\',4326)) group by NAME_0'#SQL语句
    time_start=time.time()
    cur.execute(sql)#查询
    time_end=time.time()
    rows=cur.fetchall()#拉取数据
    print('西经60度经线穿过以下国家:')
    for i in rows:#结果输出
        print(i[0])
    print('耗时:',time_end-time_start,'s')
    close_db(cur,conn)#关闭
```

4>查询结果：一共 10 个国家（如下）被西经 60 度经线穿过，耗时约 0.47s 左右（更多实验结果分析见本章第三节的《对比分析》）

```
西经60度经线穿过以下国家:
Antarctica
Argentina
Bolivia
Brazil
Canada
Falkland Islands
Greenland
Guyana
Paraguay
Venezuela
耗时: 0.47110557556152344 s
```

2. MongoDB 检索（无索引）

任务为检索西经 60 度经线穿过的国家，我采用【python】语言应用第三方库 pymongo 来解决问题。具体步骤如下（完整程序见附录 select_mongodb.py，第三方库为 pymongo 和 time）：

1>建立连接，指定数据库、集合：使用 MongoClient 建立连接，指定 test 数据库，集合为 mawenzhuo。

```
#建立连接，指定数据库集合
client=MongoClient('localhost',27017)
db=client.test
collection=db.mawenzhuo
```

2>进行查询：使用 find 方法对数据中心多边形和西经 60 度经线（即[[-60,-89.99],[-60,89.99]]）求交集，得到结果（结果仅投影出 properties）。

```
#进行查询
time_start=time.time()
ans=collection.find({
                    "geometry":{"$geoIntersects":
                    {"$geometry":{"type":"LineString","coordinates":
                    [[-60,-89.99],[-60,89.99]]}}},
                    {"_id":0,"properties":1})
```

3>结果打印：通过遍历，去掉重复的国家，打印结果。

```
#结果打印
print('西经60度经线穿过以下国家:')
countrys=[]
for i in ans:
    property=i['properties']
    country=property['NAME_0']
    if country not in countrys:
        countrys.append(country)
for i in countrys:
    print(i)
time_end=time.time()
print('耗时:',time_end-time_start,'s')
```

4>查询结果：查询结果一共 9 个国家，比 postgis 查询少了一个南极洲（这是因为南极洲的数据大于 16M，在导入 MongoDB 时被忽略造成的）。

```
西经60度经线穿过以下国家:
Argentina
Bolivia
Brazil
Canada
Falkland Islands
Greenland
Guyana
Paraguay
Venezuela
耗时: 1178.7614204883575 s
```

3. MongDB 检索（有索引）

1>在命令行界面使用【getIndexes()】命令查看当前索引（还未创建空间索引），然后使用【creatIndex({“geometry”:”2dsphere”})】命令创建空间索引，但是出现如下问题。

```
db.mawenzhuo.getIndexes()
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
db.mawenzhuo.createIndex({"geometry":"2dsphere"})

        "ok" : 0,
        "errmsg" : "Index build failed: 315b952a-4e33-49b0-8fe0-27b0fa84c7d3: Collection test.mawenzhuo ( ff929e3f-b18e-
93e-bb61-0dda048697c1 ) :: caused by :: Can't extract geo keys: { _id: ObjectId('6277b3a01304d46c5dd51ffa'), type: \"Fe
ture\", properties: { UID: 47656.0, GID_0: \"EGY\", NAME_0: \"Egypt\", VARNAME_0: null, GID_1: \"EGY.17_1\", NAME_1: \"
swan\", VARNAME_1: \"Assouan|Assu |Assuán|Syene\", NL_NAME_1: \"□□□□□□□□□□□□□□□□\", ISO_1: null, HASC_1: \"EG.AN\", CC_1:
ull, TYPE_1: \"Muhafazah\", ENGTYPE_1: \"Governorate\", VALIDFR_1: \"Unknown\", GID_2: \"EGY.17.5_1\", NAME_2: \"Idfu\"
 VARNAME_2: null, NL_NAME_2: null, HASC_2: \"EG.AN.ED\", CC_2: null, TYPE_2: \"Markaz\", ENGTYPE_2: \"Subdivision\", VA
IDFR_2: \"Unknown\", GID_3: null, NAME_3: null, VARNAME_3: null, NL_NAME_3: null, HASC_3: null, CC_3: null, TYPE_3: nul
, ENGTYPE_3: null, VALIDFR_3: null, GID_4: null, NAME_4: null, VARNAME_4: null, CC_4: null, TYPE_4: null, ENGTYPE_4: nu
l, VALIDFR_4: null, GID_5: null, NAME_5: null, CC_5: null, TYPE_5: null, ENGTYPE_5: null, GOVERNEDBY: null, SOVEREIGN:
"Egypt\", DISPUTEDBY: null, REGION: null, VARREGION: null }, geometry: { type: \"MultiPolygon\", coordinates: [ [ [ [ 3
```

2>经过分析查阅资料，认为是由于当多边形中的 2 个节点靠近在一起，甚至是重复时，似乎会发生这种情况。于是，使用先建立索引，在导入数据（导入数据过程中使用异常抛出机制自动剔除无法建立索引的数据，这样就不用手动检查剔除了）。

首先使用【drop】命令删除集合 mawenzhuo

```
> db.mawenzhuo.drop()
true
```

使用【createColection】命令在创建 mawenzhuo 集合

```
> db.createCollection('mawenzhuo')
{ "ok" : 1 }
```

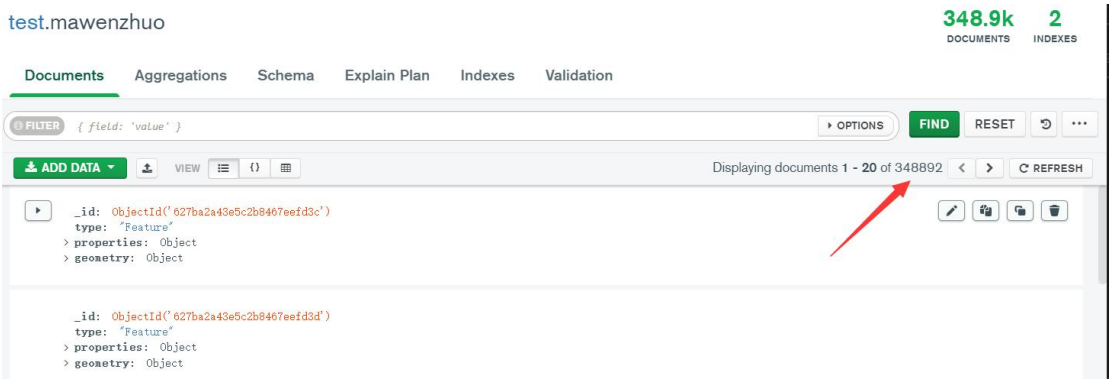
使用【creatIndex({“geometry”:”2dsphere”})】命令创建空间索引成功

```
> db.mawenzhuo.createIndex({"geometry":"2dsphere"})
{
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "createdCollectionAutomatically" : false,
        "ok" : 1
}
```

在之前的第一章第三节的 load_data 函数中，在插入数据部分加入异常捕捉和异常处理，自动筛选出异常数据，然后进行数据导入。根据导入结果可以看到，有 9 条数据因为建立空间索引异常被抛出。

```
try:
    collection.insert(each_data)#插入数据库集合
except Exception as e:
    print('ERRO'+str(index)+':')
    print(e)
```

```
> db.mawenzhuo.getIndexes()
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "geometry" : "2dsphere"
                },
                "name" : "geometry_2dsphere",
                "2dsphereIndexVersion" : 3
        }
]
```



3>进行查询：依旧使用之前编写的 select_mongodb 函数进行查询西经 60 度经线穿过的国家。结果与之前一致，速度大约快了 170 倍。（可以发现虽然结果一致，但是查询结果顺序与无索引的时候不同，这是因为建立空间索引之后，查询按照空间索引进行，速度会快很多，顺序自然就不同了）。

```
西经60度经线穿过以下国家:
Venezuela
Brazil
Guyana
Paraguay
Bolivia
Argentina
Falkland Islands
Canada
Greenland
耗时: 7.475087881088257 s
```

4. 对比分析

将 postgis、mongdb（无索引）、mongodb（有索引）的情况各做 5 次试验，查询时间如下表所示。我们可以得出结论，查询耗时：postgis<mongdb（有索引）<mongdb（无索引）。并且有索引 mongdb 耗时大约为 postgis 的 15 倍，无索引 mongdb 耗时大约为 postgis 的 2520 倍。

很明显mongodb索引建立前后的查询速度差异是由于空间索引的建立引起的。而postGis快于Mongdb查询主要是因为postgis使用的是postgres关系型数据库(其为结构严谨的表结构)再加上postgis模块的空间处理能力,而mongdb为结构较为松散的非关系型数据库，从而导致了postGis的空间事务能力强于mongdb。

实验序号\耗时（s）	PostGis	MongDB（无索引）	MongDB（有索引）
1	0.4711	1178.76	7.4751
2	0.4761	1240.53	6.9817
3	0.4681	1190.11	7.2405
4	0.4981	1204.65	7.0764
5	0.4740	1171.30	6.8992
Avg	0.4775	1197.07	7.1346

三、统计、数据处理函数实现

----------任务及要求----------

任务：利用程序，基于postgis数据库现如下功能并进行测试（40分）

要求：

①实现一个函数，输入任意一个国家，统计出这个国家的县级行政区的数量（即一个国家数据中包含的要素数量），并统计这个国家中县级行政区面积最大的三个，打印出名字和面积值（10分）；

②实现一个函数，输入任意一个国家，统计出这个国家在地域上邻接的其它国家（需要做数据的综合，因为只有县级数据）（10分）；

③实现一个数据处理函数，函数接受三个参数：国家名，像素分辨率，输出目录。输入任意一个国家，将这个国家的数据按指定像素分辨率进行渲染（像素分辨率即一个像素代表的实际地理范围的大小，宽和高等分辨率），然后按256x256像素大小的索引格网，对指定国家的渲染后栅格进行切分，切成多个正方形的影像数据并写入到输出目录中（这个过程也称为切片）。需要考虑对应国家的最大外接矩形边界，然后按比例生成图片，每个小的行政区域随机指定一种颜色，所有县级边界用黑线隔开。输出切片影像数据的命名采用索引方式，例如，最左上角的切片文件名为1-1，其下的为2-1，右侧的为1-2，以此类推。如果输出数据格式为jpg等不带有坐标信息的栅格数据，则根据每个切片文件的数据坐标参数，生成对应的坐标配准文件，例如，jpg文件要输出对应的jgw文件，最终在qgis中打开全部切片数据，综合显示并截图。（20分）；

④每个函数单独实现，并且分别用不同国家测试至少2次，③中的分辨率参数用0.1度和0.2度分别测试。所有实现方法和步骤，在报告中要详细说明。

----------实施步骤----------

1. 县级数量查询

任务为查询指定国家的县级行政区数量并输出该国家面积前三的县级行政区的名称和面积，我使用【python】语言实现该功能（完整程序见附录select_num.py，使用的带第三方库为psycopg2），具体步骤如下：
1>连接数据库、关闭数据库函数编写：本步骤编写的connect_db和close_db函数与第二章第一节一致，在此就不过多赘述。

```
#连接数据库
def connect_db():
    try:
        conn=psycopg2.connect(database='test',user='postgres',
        password='090012',host='127.0.0.1',port=5432)
    except Exception as e:
        print(e)
    else:
        return conn
    return None

#关闭数据库
def close_db(cur,conn):
    conn.commit()#事物提交
    cur.close()
    conn.close()#关闭
```

2>进行查询：这里编写select_num函数，输入为国家名。首先连接数据库，生成游标对象：

```
conn=connect_db()#连接数据库
if not conn:
    return None
cur=conn.cursor()#创建游标对象
```

使用sql语句查询该国家县级区域数量:使用【COUNT(*)】统计数目,使用【where NAME_0 like country_name】语句查询指定国家。然后使用execute执行sql语句，使用fetchall拉取结果数据.

```
#sql语句查询每个国家的县级行政区数量
sql1="SELECT COUNT(*) from mawenzhuo where NAME_0 like '%"+country_name+"%'"
cur.execute(sql1)#执行查询
num=cur.fetchall()#抓取数据
```

使用sql语句查询该国家面积前三大的县级区域：使用【ST_AREA】进行面积计算（值得注意的是，ST_AREA函数适合于计算以m为单位的坐标系中的面积，但是我们这里使用的是WGS84（srid=4326），WGS84以度为坐标系。因此我在这里使用【ST_TRANSFORM(geometry, 4527)】将其转化到以m为单位的坐标系下进行计算）。使用【ORDER BY ... DESC】语句对数据按面积进行降序排列。然后使用execute执行sql语句，使用fetchall拉取结果数据.

```
#sql语句查询这个国家的县级行政区名称、面积
sql2="SELECT NAME_3,ST_AREA(ST_TRANSFORM(mawenzhuo.geom,4527)) FROM mawenzhuo \
    WHERE NAME_0 like '%"+country_name+"%' \
    ORDER BY ST_AREA(ST_TRANSFORM(mawenzhuo.geom,4527)) DESC"
cur.execute(sql2)
area=cur.fetchall()
```

最后打印结果（面积查询的前三位即为最大的三个区域），关闭数据库。

```
print(country_name+'包含县级行政区数目为:',num[0][0])
print(country_name+'中面积前三的县级行政区依次如下:')
for i in range(3):#输出前三大的区域
    print('The area of '+area[i][0]+' is '+str(area[i][1])+' m2')
close_db(cur,conn)#关闭数据库
```

3>实验结果：使用 China 和 Canada 做示例输入，结果如下。

```
请输入想要查询的国家名称（英文）：China
China包含县级行政区数目为: 2435
China中面积前三的县级行政区依次如下:
The area of Ruoqiang is 226027815260.8579 m2
The area of Nyima is 214233266254.76028 m2
The area of Qiemo is 169361614669.2584 m2
```

```
请输入想要查询的国家名称（英文）：Canada
Canada包含县级行政区数目为: 5582
Canada中面积前三的县级行政区依次如下:
The area of Baffin, Unorganized is 1025440593570.046 m2
The area of Fort Smith, Unorganized is 884265758972.0548 m2
The area of Inuvik, Unorganized is 633580217980.0795 m2
```

2. 邻接国家查询

任务为查询输入国家的邻接国家，我使用【python】语言完成任务。完整代码见附录 select_near.py，使用第三方库为 psycopg2，具体步骤如下：

1>编写连接数据库、关闭数据库函数：与第三章第一节一致，在此不过多赘述。

```
#连接数据库
def connect_db():
    try:
        conn=psycopg2.connect(database='test',user='postgres',
        password='090012',host='127.0.0.1',port=5432)
    except Exception as e:
        print(e)
    else:
        return conn
    return None

#关闭数据库
def close_db(cur,conn):
    conn.commit()#事物提交
    cur.close()
    conn.close()
```

2>编写查询函数：这里封装为 select_near，输入参数为国家名。首先使用编写的 connect_db 连接数据库，生成游标对象。

```
#查询输入国家的邻接国家
def select_near(country_name):
    conn=connect_db()#连接数据库
    cur=conn.cursor()#创建游标对象
```

使用 sql 语句进行邻接国家的查询（这里使用连接查询的思想来完成：表 2 为所查询的表，表 1 为指定国家数据组成的表。使用 ST_TOUCHES 判断两者是否邻接。具体解释如下图所示）。

```
sql="SELECT table2.NAME_0 FROM mawenzhuo table2,\ 在表2中搜索国家名
    (SELECT * FROM mawenzhuo where NAME_0 like '%"+country_name+"%') table1 \   表1为指定国家的相关县级数据
    where ST_TOUCHES(table1.geom,table2.geom) \  表1多边形和表2多边形邻接
    and table2.NAME_0 <> '"+country_name+"' \   表2的国家名不等于指定国家名
    GROUP BY table2.NAME_0"     按表2国家名分组查询
```

使用 execute 执行查询，使用 fetchall 拉取结果数据，最后进行打印，关闭数据库。

```
cur.execute(sql)#查询
countrys=cur.fetchall()#拉取数据
#打印结果
print('与'+country_name+'相邻接的国家一共'+str(len(countrys))+',具体如下:')
for i in range(len(countrys)):
    print(countrys[i][0])
close_db(cur,conn)#关闭数据库
```

3>查询结果：这里以 China 和 India 为示例，结果如下：

```
请输入想要查询的国家(英文):China
与China相邻接的国家一共22,具体如下:
Afghanistan
Aksai Chin
Arunachal Pradesh
Azad Kashmir
Bhutan
India
Jammu and Kashmir
Kaurik
Kazakhstan
Kyrgyzstan
Laos
Lapthal
Mongolia
Myanmar
Nepal
North Korea
Pa-li-chia-ssu
Russia
Sang
Shaksgam Valley
Tajikistan
Vietnam
```

```
请输入想要查询的国家(英文):India
与India相邻接的国家一共12,具体如下:
Arunachal Pradesh
Bangladesh
Bhutan
China
Jammu and Kashmir
Kaurik
Lapthal
Myanmar
Nepal
Pakistan
Sang
Sri Lanka
```

3. 任意分辨率切片

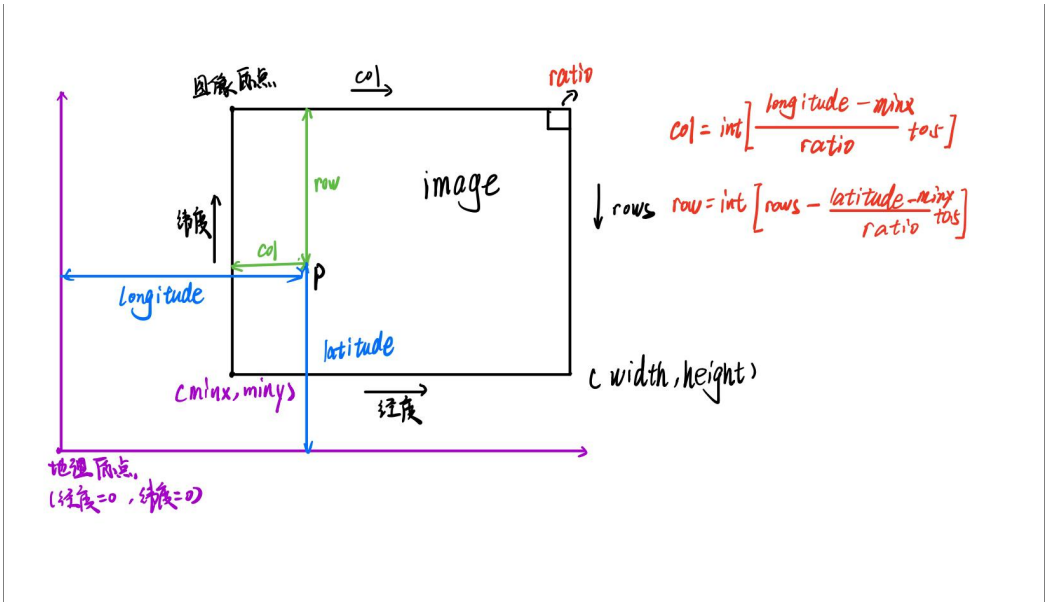
任务为对指定国家按指定分辨率进行渲染，然后切片存储，我使用【python】语言解决。代码详见附录 get_slice.py，使用的第三方库为 psycopg2、Image（PIL）、ImageDraw（PIL）、random、json。具体步骤如下：

1>连接数据库、关闭数据库函数：函数分别为 connect_db 和 close_db，与前两节一致，在此不过多赘述。

```python
#连接数据库
def connect_db():
    try:
        conn=psycopg2.connect(database='test',user='postgres',
        password='090012',host='127.0.0.1',port=5432)
    except Exception as e:
        print(e)
    else:
        return conn
    return None

#关闭数据库
def close_db(cur,conn):
    conn.commit()#事物提交
    cur.close()
    conn.close()#关闭
```

2>坐标转换函数（transform）：此函数功能为经纬坐标转像素坐标。需要明确的是，图像坐标系原点在左上角，经纬度原点则与赤道和子午线交界处，其转换如下图所示：




```python
#经纬坐标转像素坐标
def transform(longitude,latitude,width,height,minx,miny,ratio):
    #longitude,latitude经纬度坐标
    #width，height图像宽高
    #minx，miny图像左下角经纬度坐标
    #ratio分辨率
    col=int((longitude-minx)/ratio+0.5)#计算列号
    row=int(height-(latitude-miny)/ratio+0.5)#计算行号
    return row,col#返回行列号
```

3>查询函数（select）：此函数功能为根据输入的 sql 语句在 postgres 数据库中查询，返回查询结果。主要是调用前面编写的 connect_db 连接数据库，创建游标对象，使用【execute】方法进行查询，调用之前编写的 close_db 函数关闭连接，返回查询结果。

```
#查询
def select(sql):
    conn=connect_db()#连接数据库
    cur=conn.cursor()#生成游标对象
    cur.execute(sql)#执行查询
    ans=cur.fetchall()#抓取数据
    close_db(cur,conn)#关闭数据库连接
    return ans
```

4>渲染、切片函数（select_slice）：此函数为主要功能函数，可以将指定国家按照指定分辨率渲染，并且按照 256x256 的格网切片。输入参数为（国家名，分辨率，输出目录），具体步骤如下：

①得到指定国家每个县级多边形的坐标范围：使用 sql 语句调用之前编写的 select 函数进行查询，得到结果 ans。Sql 语句中使用【ST_XMIN，ST_XMAX】等空间函数得到多边形的坐标最小最大值，使用【ST_ASGEOJSON】将其转为 json 格式以便后续调用。

```
#sql语句查询指定国家每条数据的最大最小经纬坐标
sql="SELECT ST_ASGEOJSON(geom),ST_XMIN(geom),ST_XMAX(geom),ST_YMIN(geom),ST_YMAX(geom)\
    FROM mawenzhuo WHERE NAME_0 like '%"+country_name+"%'"
ans=select(sql)
```

②确定外接矩形的大小（坐标范围）：实际上这是一个搜索坐标最大最小值的过程。首先定义横坐标（经度）的 max=-180，min=180，然后循环遍历之前得到的 ans 中的多边形坐标范围，如果其横坐标最小值小于 min 则更新 min，横坐标最大值大于 max 则更新 max，纵坐标（纬度）同理。同时在循环时将个多边形点坐标加入到 points_list 中以供后面绘图（这里需要使用 JSONDecoder 解码，如果不解码会出现报错，因为 json 格式中其类型不是数字而是字符串）。

```
#确定外接矩形大小
#定义初始坐标最大最小值
minx=180
maxx=-180
miny=90
maxy=-90
#循环更新边界范围,并存储边界点坐标
points_list=[]#存储经纬坐标
for i in ans:#循环更新范围
    if i[1]<minx:
        minx=i[1]
    if i[2]>maxx:
        maxx=i[2]
    if i[3]<miny:
        miny=i[3]
    if i[4]>maxy:
        maxy=i[4]
    #存储边界坐标
    temp = json.JSONDecoder().decode(i[0])#解码
    points_list.append(temp["coordinates"][0][0])#存储
print('经纬度范围:',minx,maxx,miny,maxy)
```

③确定图像的大小，生成图像：根据之前的经纬度范围、分辨率，根据如下公式可以求出图像的宽和高，然后使用 Image.new 创建图像（为了防止后面切片显示

时出现黑底，创建图片时适当增加长宽，并不影响后面的坐标转换）。

$$\text{width} = \text{int}(\frac{\max x - \min x}{ratio})$$

$$height = \text{int}(\frac{\max y - \min y}{ratio})$$

```
#根据边界范围确定图片大小
width=int((maxx-minx)/ratio)
height=int((maxy-miny)/ratio)
#创建图片
img=Image.new("RGB",(width+160,height+160),"white")
```

④转像素坐标：将之前的存储的边界点的坐标转为像素坐标。遍历每个点，对每个点调用之前编写的 transform 函数，得到其对应的像素坐标（行号对应纬度，列号对应经度）。使用 ImageDraw.Draw(img).polygon 绘制多边形（边界用黑线，内部颜色随机，为了防止出现白色，灰度值上限设为 235）。最后保存图片。

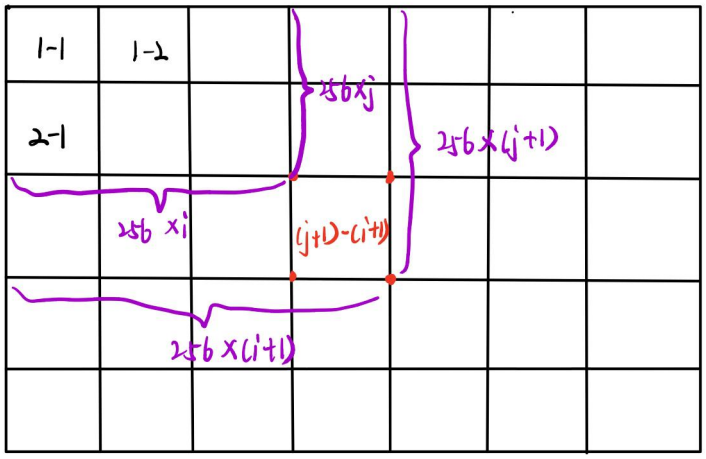
```
#转换为像素坐标
for points in points_list:
    pixels=[]#存储像素坐标
    for point in points:
        #坐标转换
        py,px=transform(point[0],point[1],width,height,minx,miny,ratio)
        pixels.append((px,py))#加入像素点
    #绘制多边形
    ImageDraw.Draw(img).polygon(pixels,#逐点绘制
                        outline='rgb(0,0,0)',#用黑色连线
                        fill=f'''rgb({random.randint(0,235)},{random.randint(0,235)},{random.randint(0,235)})''')#随机颜色填充
img.save(target_path+'result2.jpg')
```

⑤切片：

首先得到格网数量（格网大小为 256x256），为了防止边界处的切片因为四舍五入的关系丢失，所以+1.

```
#进行切片
num_x=int(width/256)+1#按照256x256的网格进行切片
num_y=int(height/256)+1
```

然后遍历格网进行切片，其中使用 crop 函数进行剪裁，每个切片的坐标范围转换公式如下。完成后按照如下图的命名规则存入对应的 jpg 文件。




```
for i in range(num_x):
    for j in range(num_y):
        img_slice=img.crop((256*i,256*j,256*(i+1),256*(j+1)))#按格网切分
        img_slice.save(target_path+'slice\\'+str(j+1)+'-'+str(i+1)+'.jpg')#存储
```

⑥生成 jgw 文件：jgw 文件格式如下图所示，使用 open 新建对应 jgw 文件，按照格式写入对应信息。

jgw文件格式

A = x 比例因子；像素的 x 方向尺寸，采用地图单位

(x-scale; dimension of a pixel in map units in x direction)

D = 旋转项,一般取0.000000(rotation terms)

B = 旋转项,一般取0.000000(rotation terms)

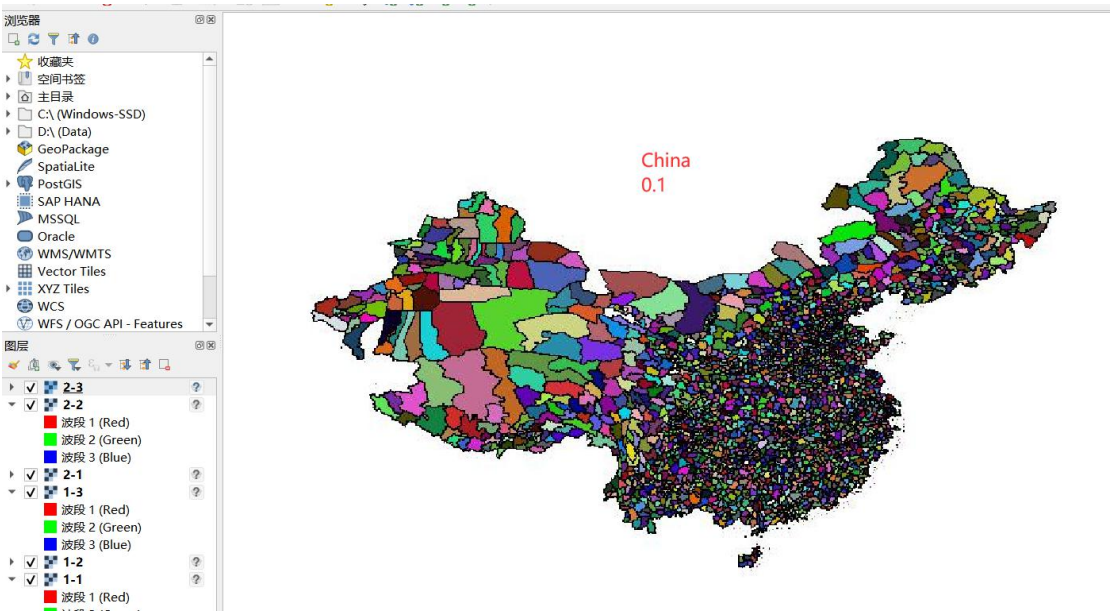
E = y 比例因子的负值；像素的 y 方向尺寸，采用地图单位(the negative of y-scale; dimension of a pixel in map units in y direction)

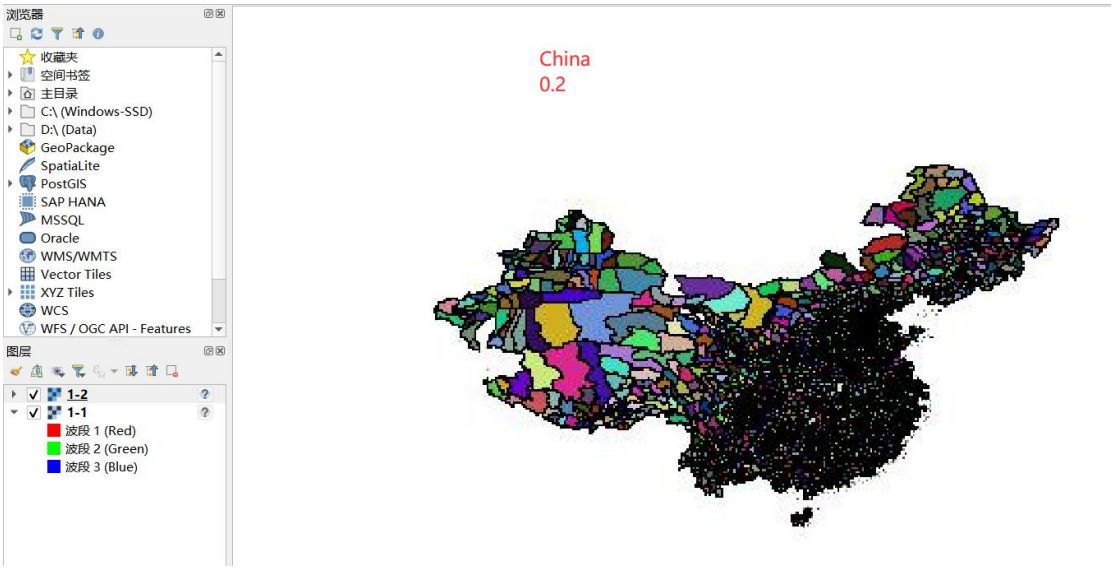
C = 平移项；左上角像素的中心点的 x地图坐标(translation terms)

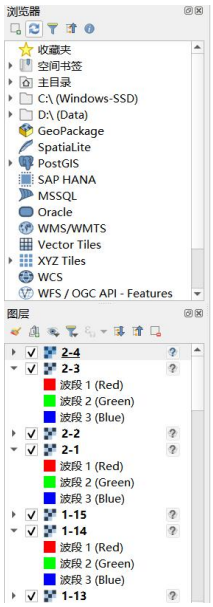
F = 平移项；左上角像素的中心点的 y 地图坐标(translation terms)

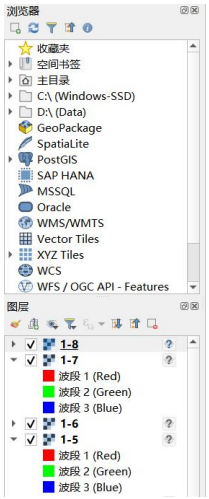
```
for i in range(num_x):
    for j in range(num_y):
        img_slice=img.crop((256*i,256*j,256*(i+1),256*(j+1)))#按格网切分
        img_slice.save(target_path+'slice\\'+str(j+1)+'-'+str(i+1)+'.jpg')#存储
        #生成对应的jgw文件
        with open(target_path + 'slice\\'+str(j+1)+'-'+str(i+1)+'.jgw','w+', encoding='UTF-8') as jgw:
            jgw.write(f'{ratio}\n'#x方向比例因子
                      f'{0}\n'#旋转项
                      f'{0}\n'#旋转项
                      f'{-ratio}\n'#y方向比例因子
                      f'{minx +256 * i * ratio}\n'#左上角经度
                      f'{maxy - 256 * j * ratio}')#左上角维度
```

⑦功能测试：我们以 China、Russia 国家，分辨率 0.1 度和 0.1 度为例，在 QGis 中显示如下。









四、Geohash 编码生成及查询实现

----------任务及要求----------

任务：利用程序，基于 mongodb 数据库实现如下功能并测试（20 分）

要求：

①实现一个函数，给定一个经纬度坐标及编码长度，则生成一个 geohash 编码（10 分）；

②实现一个函数，给定一个 mongodb 的县级行政区名字作为查询条件，即可查询出某个县级行政区域，然后，将这个县级行政区域的几何边界用 8 位 geohash 编码数据组织成为一大段文本进行描述（相邻顶点之间不留空格，连续存放编码）（5 分）；

③测试程序调用上述两个功能函数，并用“湖北省鄂州市鄂城市”的数据进行测试，打印出结果（注意：用属性从省级定位到县级行政区需要三级，本题中，相当于找 湖北省->鄂州市->鄂城市）（5 分）。

----------实施步骤----------

1. 生成 geohash 编码

为了得到给定经纬度的给定长度的 geohash 编码，我使用【python】语言解决该问题。本程序（代码详见附录 geohash.py，第三方库为 math）主要由 5 个函数组成:divide、binary2decimalism、get_binary、get_base32、get_geohash 具体步骤如下:

1>divide 函数：此函数功能为根据输入的最大最小值以及精度，得到输入数据的二进制编码。思路是将 data 与中间值 mid 比较，如果大于（等于）赋 1，否则赋 0，更新最大最小值、重新计算中值、再比较，直到达到精度为止。

```python
#二分
def divide(min,max,data,precision):
    length=0#记录长度
    code=[]#记录编码
    while length<precision:#没有达到精度前循环划分
        mid=(min+max)/2#计算中间值
        if data>=mid:#如果大于等于中间值赋1
            code.append(1)
            length=length+1
            min=mid#更新最小值
        else:#小于中间值赋0
            code.append(0)
            length=length+1
            max=mid#更新最大值
    return code#返回编码
```

2>binary2decimalism 函数：此函数的功能为将二进制（列表）转化为十进制。（这里注意，遍历二进制列表要从后向前遍历）。

```python
#二进制转十进制
def binary2decimalism(binary):
    decimalism=0
    for i in range(len(binary)-1,-1,-1):
        decimalism=decimalism+binary[i]*2**(len(binary)-i-1)
    return decimalism
```

3>get_binary 函数：此函数的功能为获取经纬度坐标的合并二进制编码。首先调用 divide 函数得到给定经纬度相应精度的二进制编码（这里的输入的精度为最后的编码长度，因此经度的二进制长度为 5*precision/2 向上取整，纬度的二进制长度为 5*precision/2 向下取整）。然后对这两个编码进行合并，原则为经度占偶数位，纬度占奇数位（从 0 开始），由于经度的二进制长度可能为奇数，因此判断是否为奇数，是的话补上最后一位。最后返回合并的二进制编码。

```python
#获取经纬度二进制编码
def get_binary(longitude,latitude,precision):
    longitude_binary=divide(-180,180,longitude,math.ceil(5*precision/2))#计算经度二进制编码
    latitude_binary=divide(-90,90,latitude,5*precision//2)#计算维度二进制编码
    code_binary=[]#存储合并的二进制编码
    for i in range(5*precision//2):#合并
        code_binary.append(longitude_binary[i])#经度占偶数位
        code_binary.append(latitude_binary[i])#纬度占奇数位
    if (precision%2)==1:
        code_binary.append(longitude_binary[-1])
    return code_binary
```

4>get_base32 函数：此函数功能为将二进制编码转化为 base32 编码。首先对二进制编码划分为 5 个一组的列表（不足 5 位补 0），然后调用 binary2decimalism 函数将每组编码转为 10 进制。对照 base32 编码表，得到最后的 base32 编码，并返回。

```
#获取base32编码
def get_base32(code_binary):
    #Base32编码表
    base32='0123456789bcdefghjkmnpqrstuvwxyz'
    #将二进制编码划分为5个一组的mini_batch
    mini_batches=[code_binary[k:k+5] for k in range(0,len(code_binary),5)]
    if len(mini_batches[-1])<5:#最后一个分组不足5个则补0
        mini_batches[-1]=mini_batches[-1]+(5-len(mini_batches[-1]))*[0]
    #对每个mini_batch计算十进制
    decimalism=[binary2decimalism(i) for i in mini_batches]
    #转换为base32编码
    code_base32=[base32[decimalism[i]] for i in range(len(decimalism))]
    str_base32=''.join(code_base32)
    return str_base32
```

5>get_geohash 函数：此函数为整体的功能函数，输入参数为（经度，纬度，精度），返回值为对应的 geohash 编码。

```
#获取geohash
def get_geohash(longitude,latitude,precision):
    code_binary=get_binary(longitude,latitude,precision)#获取二进制编码
    geohash=get_base32(code_binary)#获取base32编码
    return geohash
```

6>函数功能测试：这里我们输入（116.389550,39.928167,4）得到如下结果

```
请输入经度:116.389550
请输入纬度:39.928167
请输入精度:4
geohash编码为: wx4g
```

2. 获取县级行政区边界 geohash 编码

任务要求为获取指定县级行政区边界的 geohash 编码，我使用【python】语言解决（程序详见附录 select_geohash.py，第三方库为 pymongo 和 geohash，注意这里的 geohash 即为前一小节所编写的 geohash.py），程序主题为一个 select_geohash 函数，具体步骤如下：

1>连接 mongodb 数据库，指定集合：

```
#建立连接指定数据库集合
client=MongoClient('localhost',27017)
db=client.test
collection=db.mawenzhuo
```

2>处理输入数据：使用【split】方法将输入名称转换为各级名称组成的列表。

```
#处理输入
names=names.split("-")
```

```
China-Hubei-Ezhou-Echeng Shi
['China', 'Hubei', 'Ezhou', 'Echeng Shi']
```

3>查询指定的县级数据：查询条件以字典的形式给出，国家级、省级、县级均为查询条件。投影出 properties 和 geometry 以供后续使用。使用【find_one】方法进行查询。

```
#查询指定的县级区域
condition={}#查询条件
for i in range(len(names)):#每一级行政区名字都要吻合
    condition["properties.NAME_"+str(i)]=names[i]
ans=collection.find_one(condition,{"_id":0,"properties":1,"geometry":1})
```

4>得到 geohash 编码:对查询得到的顶点信息(即 ans[“geometry”][“coordinates”][0])调用 geohash 模块的 get_geohash 方法得到 8 位 geohash 编码，串联起来，返回结果。

```
#得到geohash编码
code=''
for point in ans["geometry"]["coordinates"][0]:
    #调用自己编写的geohash模块函数,获取每个点的8位geohash编码
    code=code+geohash.get_geohash(point[0],point[1],8)
return code
```

5>打印输出：

```
names=input('请输入想要查询的县级行政区(如China-Hubei-Ezhou-Echeng Shi):')
code=select_geohash(names)
print('geohash编码为:')
print(code)
```

6>函数测试：这里我们使用中国-湖北省-鄂州-鄂城市作为测试数据，得到结果如下（由于结果可能不太清晰，可见附录 result1.jpg）：

```
请输入想要查询的县级行政区(如China-Hubei-Ezhou-Echeng Shi):China-Hubei-Ezhou-Echeng Shi
geohash编码为:
wt3v10prwt3v10c3wt3v0bkmwt3tp0hnwt3tn1x2wt3tjg2bwt3tjqfpwt3tmh1pwt3tmh2wwt3tkypzwt3ttp4zwt3tv140wt3tuum6wt3tuvcbwt3tuyhzwt3tuyx5wt3tvnbywt3tvp4mwt3tvp78wt3tvpmk
wt3tvpy6wt3wj0rjwt3wj0zrwt3wj1wbwt3wj4n8wt3wj62nwt3wj750wt3wj7mtwt3wjebfwt3wjs78wt3wjstfwt3wjubswt3wjv5hwt3wjvm8wt3wjvrewt3wnj9bwt3wnjefwt3wnjtfwt3wnjpywt3wnkc0
wt3wnk6ewt3wn7guwt3wn7kzwt3wn7n4wt3wn6xtwt3wnd34wt3wnd4uwt3wndjuwt3wnf0jwt3wnf1twt3wnf5qwt3wnfhuwt3wncvwwt3wnctywt3wncq7wt3wnby8wt3wnbnuwt3tyzxqwt3tyzr3wt3tyyyy
wt3tyyw6wt3tyynjwt3tyvvdwt3tyvmhwt3tyvj1wt3tyutmwt3tyumtwt3tyunpwt3tyupjwt3tzh30wt3tzh75wt3tzhmhwt3tzhnzwt3tz7bzwt3tz7drwt3tz7sbwt3tz7x2wt3tze3twt3tze7swt3tzemk
wt3tzerjwt3tzg8jwt3tzgbzwt3tzu3bwt3tzue8wt3tzuv2wt3tzuz3wt3vbhc4wt3vbhguwt3vbjnjwt3vbm2wwt3vbmf3wt3vbqhmwt3vbqw0wt3vbqzrwt3vbx2mwt3vbxc0wt3y084xwt3y08u0wt3y09ms
wt3y09vfwt3y0dnpwt3y0dw9wt3y0dztwt3y0g1kwt3y0g7twt3y0gy9wt3y15cpwt3y1h74wt3y1hgcwt3y1jk7wt3y1ju7wt3y1nm1wt3y1nxjwt3y1r42wt3y1rtfwt3y32pcwt3y389qwt3y394zwt3y39eg
wt3y3dj7wt3y3dw9wt3y3fbhwt3y3g1fwt3y3fgpwt3y3fvkwt3y3fwbwt3y3fpdwt3y618rwt3y613swt3y6157wt3y60vfwt3y622qwt3y6247wt3y4rgbwt3y4rhvwt3y4qtkwt3y4qj6wt3y4mtswt3y4krv
wt3y4kprwt3y46c8wt3y43brwt3y40whwt3vf5uxwt3vf5m3wt3vf1kzwt3vf0epwt3v9zzxwt3vcb30wt3vc8umwt3vc91hwt3vc2ufwt3vc20hwt3v9nt7wt3v9jmbwt3v9hkbwt3v91skwt3v3p2uwt3v2yqy
wt3v2uddwt3v29zqwt3v29xtwt3v29ptwt3v0xpuwt3v0uu6wt3v0usywt3v158nwt3v15fhwt3v1hg0wt3v1jt0wt3v1jq7wt3v15yywt3v13c5wt3v1344wt3v10pr
```

五、总结

经过本次课程设计，让我真正认识到了一点--当数据达到海量时，再简单的操作都变的复杂。以前我们接触到的数据一般最大就为几百 MB，但是本次实习数据以 GB 为单位。这不仅给硬件提出了更高的要求，也给算法等方面提出了要求。面对海量的数据，我们需要在算法上优化，使用相同的硬件设备达到最快的运行速度，同时我们还要考虑时间成本，当一次实验需要 20-30min 时，我们必须每一次实验前都再三检查，把试错成本降到最低。我想这是我第一次接触到“大数据”，其中积累的些许经验在以后面对真正的大数据时，可以更加游刃有余的面对。

其他操作过程在前面的章节有详细描述，在这里就不过多赘述。最后感谢

老师的教导，致此。