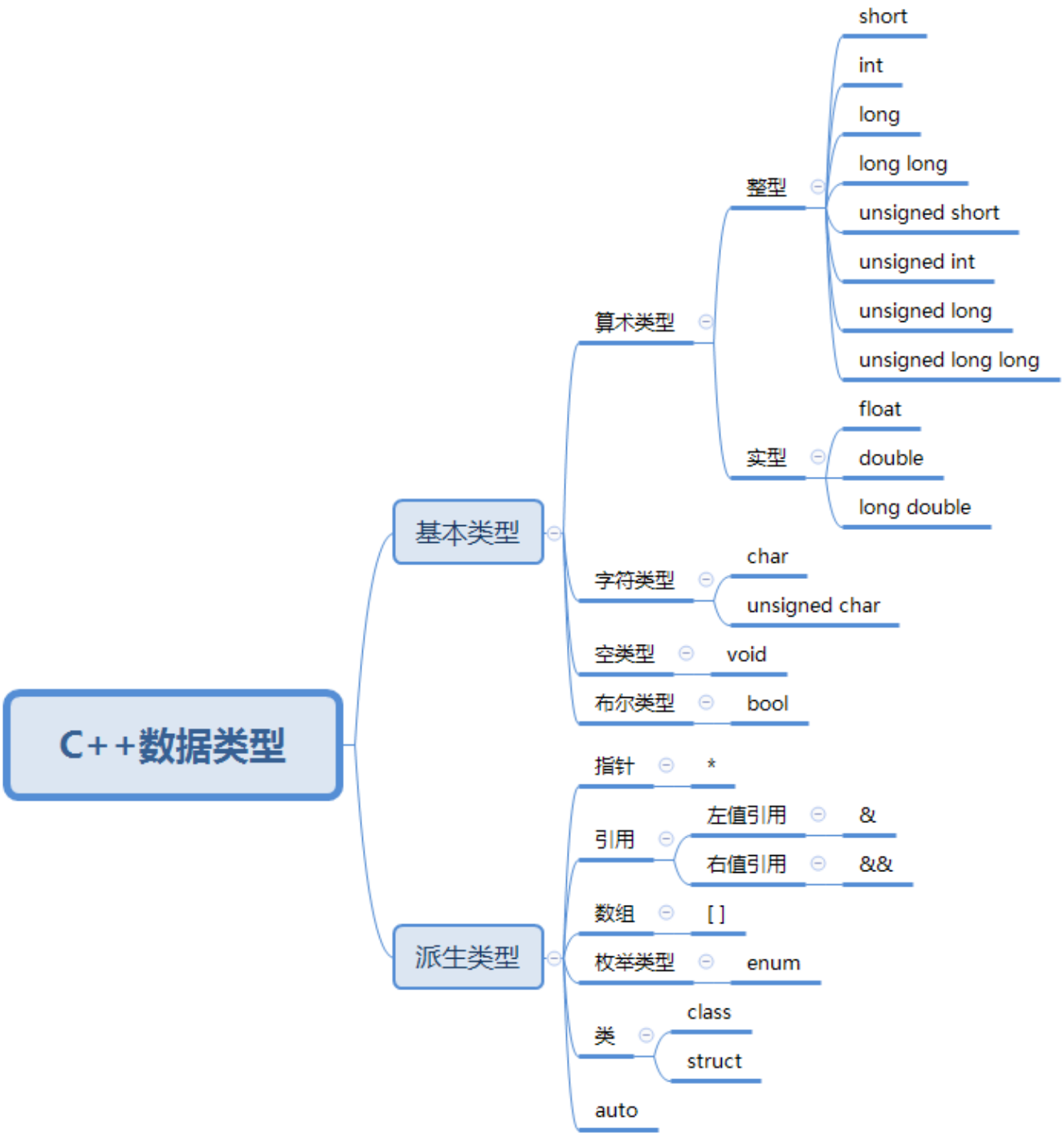
第八次作业报告

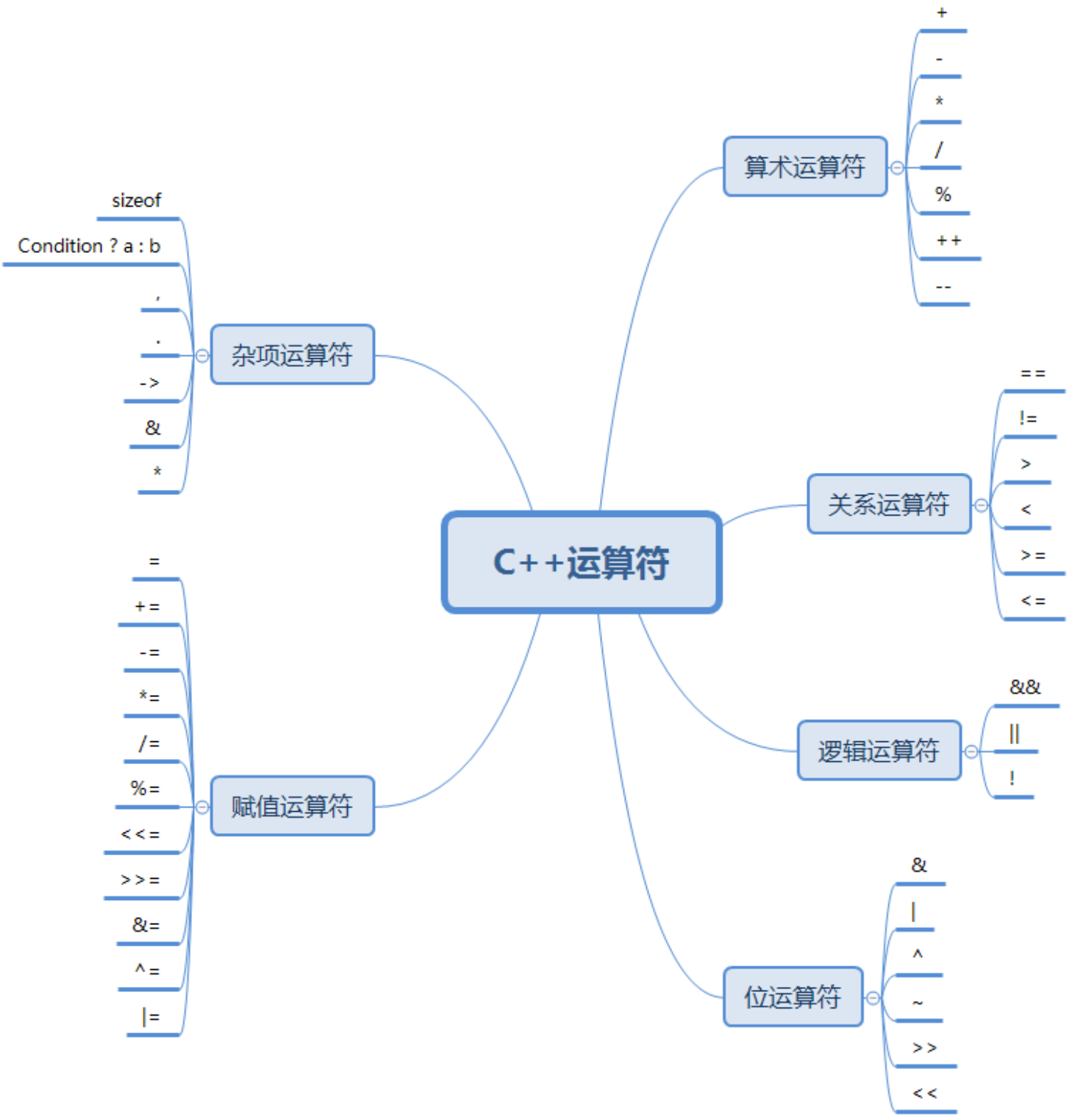
161271029 岳翔

1．C++编程语言基本成分

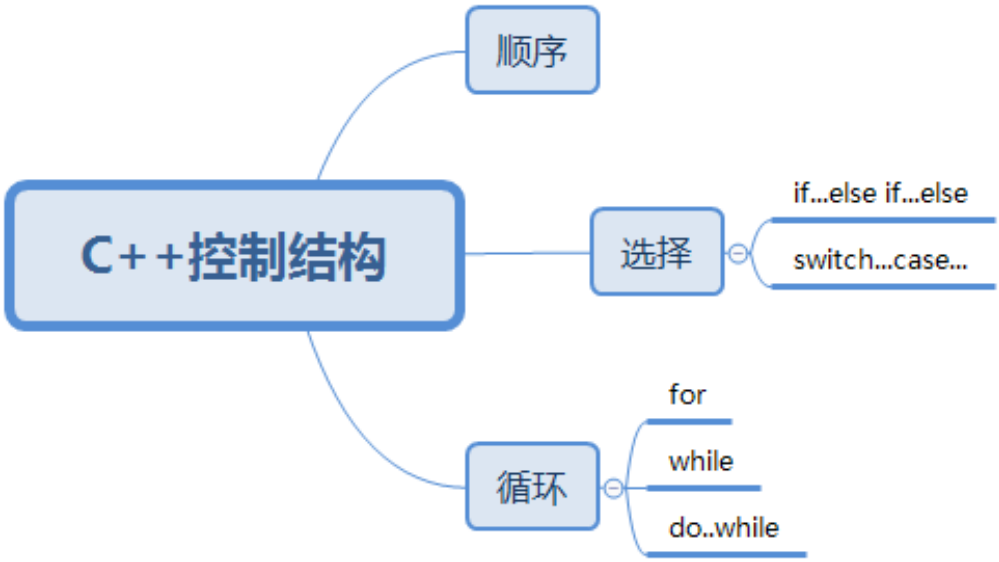
数据成分



运算成分


C++运算符
算术运算符
+
-
*
/
%
++
--
关系运算符
==
!=
>
<
>=
<=
逻辑运算符
&&
||
!
位运算符
&
|
^
~
>>
<<
赋值运算符
=
+=
-=
*=
/=
%=
<<=
>>=
&=
^=
|=
杂项运算符
sizeof
Condition ? a : b
,
.
->
&
*

控制成分



传输成分

c 风格标准输入：scanf

c 风格标准输出：printf

c 风格文件输入：fscanf

c 风格文件输出：fprinf

标准输入流：std::cin

标准输出流：std::cout

文件输入流：ifstream

文件输出流：ofstream

标准错误流：std::cerr

标准日志流：std::clog

2．C++编程语言特性

1. 语言设计特性

过程化程序设计

抽象数据类型

面向对象

继承、多态

泛型编程（模板）

2. 工程特性

静态编译

高性能计算

开发效率较低

C++标准更新（c++11、c++14....）

IDE 支持（Visual studio）

3. 应用特性

服务器端开发

底层架构

数据库

高频交易

游戏引擎

操作系统

虚拟现实

数字图像处理

科学计算

分布式应用

网络软件

设备驱动程序

3．C++开源工程 LevelDB 注释分析

1. 工程介绍

LevelDB 是 Google 开源的持久化 KV 单机数据库，具有很高的随机写，顺序读/写性能，但是随机读的性能很一般，也就是说，LevelDB 很适合应用在查询较少，而写很多的场景。LevelDB 应用了 LSM (Log Structured Merge) 策略，lsm_tree 对索引变更进行延迟及批量处理，并通过一种类似于归并排序的方式高效地将更新迁移到磁盘，降低索引插入开销，关于 LSM，本文在后面也会简单提及。

2. 注释分析

程序名	程序语言	代码行	序言性注释	功能性注释

db.h	C++	25-27	// Abstract handle to particular state of a DB.
			// A Snapshot is an immutable object and can therefore be safely
			// accessed from multiple threads without any external synchronization.	

db.h	C++	33	// A range of keys	
db.h	C++	35-36		// Included in the range // Not included in the range

db.h	C++	42-44	// A DB is a persistent ordered map from keys to values.
			// A DB is safe for concurrent access from multiple threads without
			// any external synchronization.	

db.h	C++	47-51	// Open the database with the specified "name".
			// Stores a pointer to a heap-allocated database in *dbptr and returns
			// OK on success.
			// Stores nullptr in *dbptr and returns a non-OK status on error.
			// Caller should delete *dbptr when it is no longer needed.	

db.h	C++	63-65	// Set the database entry for "key" to "value". Returns OK on success,
			// and a non-OK status on error.
			// Note: consider setting options.sync = true.	

db_impl.cc	C++	1483-1484	// Default implementations of convenience methods that subclasses of DB // can call if they wish	
write_batch.cc	C++	102	// WriteBatch header has an 8-byte sequence number followed by a 4-byte count.	
db_iter.cc	C++	209-210		// iter_ is pointing at the current entry. Scan backwards until // the key changes so we can use the normal reverse scanning code.
db_iter.cc	C++	242		// We encountered a non-deleted value in entries for previous keys,

风格：

内部文档：

- 说明文档齐全，各个文件夹下有.md 文件作说明。
- github 上的 README.md 也很详细。

数据说明：

- 用户的主要使用类为 DB 类，因此 DB 类的声明与实例文件中作者注释齐全。在代码中可以看到 DB 类中的每一个方法都有序言式注释作说明，而且针对复杂结构有嵌入的功能性注释。
- 注释非常齐全，一行模块声明代码对应 4-8 行序言式注释。
- 各个变量名命名易读易懂。
- db.h 大多都是声明，因此实际代码量很少，具体实现写在了 db_impl.cc 中。声明和实现分离是很值得借鉴的代码风格。
- 进行了类的高度封装，使得代码呈现时没有太多的临时变量，简洁易懂。

语句构造：

- 大括号采用 K&R 风格。
- 作者的类方法在引入两个以上的参数，或者参数类型较长的情况下，会采用一个参数一行的方式书写，大大增加了代码可读性。
- 循环嵌套不超过三次。
- 二元运算符的两侧有空格，代码书写较为宽松，阅读流畅。
- 在类的结束行，加入一条类名提示的注释，方便用户在模块尾部看到模块名。

3. 个人项目代码修改

由于本人项目主要的交互逻辑位于 views.py，因此更改如下：

更改前：

```
from django.shortcuts import render, redirect, HttpResponse
from django.db import connection
from django.views.decorators.csrf import csrf_exempt
from mysite.articleStruct import *


def index(req):
    ctx = {}
    c = connection.cursor()
    c.execute('select * from articles')
    ctx['articles'] = c.fetchall()
    c.execute('select * from sider')
    ctx['sider'] = c.fetchall()
    ctx['imgs'] = list(range(1,4))
    return render(req, 'index.html', ctx)

@csrf_exempt
def articles(req):
    ctx = {}
    c = connection.cursor()
    c.execute('select * from articles order by time desc')
    Arts = []
    vec_art = c.fetchall()
    for it_art in vec_art:
        siderName = 'sider-art-%s' % it_art[0]
        c.execute('select * from `%s` order by time desc', siderName)
        vec_comm = c.fetchall()
        Comms = []
        for it_comm in vec_comm:
            Comms.append(Comm(it_comm[0], it_comm[1], it_comm[2],
it_comm[3]))
        Arts.append(Art(it_art[0], it_art[1], Comms))
    ctx['Arts'] = Arts

    if req.is_ajax():
        # print(req.body)
        if req.POST.get('crateArt'):
            mdHTML = req.POST['mdHTML']
            timestamp = req.POST['timestamp']
            c.execute('insert into articles(time, content)
values(%s, %s)', (timestamp, mdHTML))
            siderName = 'sider-art-%s' % timestamp
            c.execute('CREATE TABLE `%s` (  `time` VARCHAR(30) NOT NULL,
`content` VARCHAR(100) NOT NULL,  `link` VARCHAR(30) NOT NULL,  `author`
VARCHAR(30) NOT NULL,  PRIMARY KEY (`time`))', siderName)

        if req.POST.get('delArt'):
            c.execute('DELETE FROM `articles` WHERE `time`=%s',
req.POST['time'])
            siderName = 'sider-art-%s' % req.POST['time']
            c.execute('DROP TABLE `%s`', siderName)

        if req.POST.get('addComm'):
            time = req.POST['timestamp']
            link = req.POST['link']
            content = req.POST['content']
```

```
            author = req.POST['author']
            siderName = 'sider-art-%s' % link
            c.execute('insert into `%s` values(%s, %s, %s, %s)',
(siderName, time, content, link, author))

    return render(req, 'articles.html', ctx)
```

更改后:

```
from django.shortcuts import render, redirect, HttpResponse
from django.db import connection
from django.views.decorators.csrf import csrf_exempt
from mysite.articleStruct import *

# index.html 视图函数
# 从数据库中取出文章和侧边栏内容进行循环渲染
def index(req):
    ctx = {}
    c = connection.cursor()
    c.execute('select * from articles') # SQL: 取出文章
    ctx['articles'] = c.fetchall()
    c.execute('select * from sider') # SQL: 取出侧边栏内容
    ctx['sider'] = c.fetchall()
    ctx['imgs'] = list(range(1,4)) # 图片名列表[1, 2, 3]
    return render(req, 'index.html', ctx) # index(req)


# articles.html 视图函数
# 从数据库中取出文章和文章对应评论表进行循环渲染
# 并根据 ajax 返回数据进行博文的发表、删除与评论的发表操作
@csrf_exempt
def articles(req):
    ctx = {}
    c = connection.cursor()
    c.execute('select * from articles order by time desc') # SQL: 取出文
章
    Arts = []
    vec_art = c.fetchall()
    # 对于每一篇文章进行循环
    for it_art in vec_art:
        siderName = 'sider-art-%s' % it_art[0]
        c.execute('select * from `%s` order by time desc', siderName) #
SQL: 取出文章对应的评论表
        vec_comm = c.fetchall()
        Comms = []
        # 对评论表中的每一条评论，都添加到 Comms 列表
        for it_comm in vec_comm:
            Comms.append(Comm(it_comm[0], it_comm[1], it_comm[2],
it_comm[3]))
        # 向 Arts 列表添加一个 Art 对象
        Arts.append(Art(it_art[0], it_art[1], Comms))
    ctx['Arts'] = Arts

    # 判断收到 ajax 异步请求
    if req.is_ajax():

        # 创建文章
        if req.POST.get('crateArt'):
            mdHTML = req.POST['mdHTML']
            timestamp = req.POST['timestamp']
            c.execute('insert into articles(time, content)
values(%s, %s)', (timestamp, mdHTML))
```

```
            siderName = 'sider-art-%s' % timestamp
            # 创建文章表SQL 语句
            c.execute('CREATE TABLE `%s` (  `time` VARCHAR(30) NOT NULL,
`content` VARCHAR(100) NOT NULL,  `link` VARCHAR(30) NOT NULL,  `author`
VARCHAR(30) NOT NULL,  PRIMARY KEY (`time`))', siderName)

        # 删除文章
        if req.POST.get('delArt'):
            # SQL: 删除文章表
            c.execute('DELETE FROM `articles` WHERE `time`=%s',
req.POST['time'])
            siderName = 'sider-art-%s' % req.POST['time']
            # SQL: 删除文章对应评论表
            c.execute('DROP TABLE `%s`', siderName)

        # 增加评论
        if req.POST.get('addComm'):
            time = req.POST['timestamp']
            link = req.POST['link']
            content = req.POST['content']
            author = req.POST['author']
            siderName = 'sider-art-%s' % link
            # SQL: 向评论表插入评论
            c.execute('insert into `%s` values(%s, %s, %s, %s)',
(siderName, time, content, link, author))

    return render(req, 'articles.html', ctx) # articles(req)
```