w



**QG 工作室项目报告**

学 院 物理与光电工程学院

专 业 光电信息科学与工程

班 级 2023级 1 班

组 别 图形组

姓 名 李文皓

学 号 3123006526

### 2024 年 4 月

### 目录

目录

[2024 年 4 月 1](#_Toc164896528)

[目录 2](#_Toc164896529)

[一．项目简介 3](#_Toc164896530)

[1.1 设计要求 3](#_Toc164896531)

[1.2 功能模块划分 3](#_Toc164896532)

[二．技术原理 6](#_Toc164896533)

[2.1 技术需求分析 6](#_Toc164896534)

[2.2 确定技术选型和外部库 6](#_Toc164896535)

[三．功能模块详细设计 7](#_Toc164896536)

[3.1 地形生成模块 7](#_Toc164896537)

[3.2 昼夜更替模块 11](#_Toc164896538)

[3.3 特效模块 13](#_Toc164896539)

[3.4 实体控制模块 15](#_Toc164896540)

[3.5 摄像机模块 16](#_Toc164896541)

[3.6 放置与破坏模块 13](#_Toc164896539)

[3.7 ImGui模块 13](#_Toc164896539)

[四．程序测试 20](#_Toc164896542)

[4.1 测试列表 20](#_Toc164896543)

[4.2 测试截图 21](#_Toc164896544)

[五．程序亮点 31](#_Toc164896545)

[5.1 性能优化 31](#_Toc164896546)

[5.2 用户体验 32](#_Toc164896547)

[5.3 安全性 33](#_Toc164896548)

[六．程序难点 34](#_Toc164896549)

[1. 前端独立窗口设计难点 34](#_Toc164896550)

[2. 后台数据库表设计难点 34](#_Toc164896551)

[3. 后台数据库表设计难点 34](#_Toc164896552)

[4. 后台服务器设计难点 34](#_Toc164896553)

[5. 后台服务器设计难点 34](#_Toc164896554)

[七．体会总结 35](#_Toc164896555)

[7.1 学习与收获 35](#_Toc164896556)

[7.2 项目体会 38](#_Toc164896557)

[7.3 总结 38](#_Toc164896558)

# 一．项目简介

## 设计要求

**通过DirectX实现一个类我的世界游戏，实现地形场景生成和天空盒，第一人称和第三人称控制，昼夜更替，导入人物模型等功能。**

## 功能模块划分

**在具体的功能需求上分析的基础上，我将本软件的功能需求分为几大模块，如下图所示:**

#### 功能模块划分（大纲）

#### 

#### 具体的功能模块

##### 地形生成模块

* 1. 物块管理
  2. 区块设计
  3. 区块生成
  4. 区块绘制
  5. 地表物体生成

##### 昼夜更替模块

* 1. 天空盒
  2. 方向光

##### 特效模块

* 1. 雾效
  2. 淡入淡出
  3. 小地图

##### 实体控制模块

* 1. 玩家控制
  2. 敌人控制

##### 摄像机控制模块

* 1. 第一人称控制
  2. 第三人称控制
  3. 自由视角控制

##### 放置与破坏模块

* 1. 碰撞箱检测
  2. 射线检测

1. ImGui模块
   1. ImGui控制游戏参数
   2. ImGui控制特效选择

# 程序整体大致流程

# 二．技术原理

## 技术需求分析

**在技术原理方面， 由于游戏要追求流畅，因此需要使用实例化技术来高效的绘制网格模型相同的物体，从而减少因大量顶点和索引输入装配到渲染管线而造成大量不必要的开销。或是使用视锥体剔除技术来判断物体的碰撞箱是否在视锥体中，从而避免绘制可视范围外的物体，减少GPU的负担，但如此会增大CPU的负担，因此需要结合实际情况来使用，在我的项目里加入了视锥体剔除，但由于物块过多会给CPU造成过大的负担，因此默认关闭，可使用ImGui图形界面来打开。**

**模型方面使用了Assimp模型加载库来导入模型。还使用Render-To-Texture技术来将原本渲染后输出的纹理重新作为输入来实现全屏特效和小地图。地形方面还使用了柏林噪声算法来生成地表，该技术还可作用在X轴和Z轴上来生成洞穴。**

## 确定技术选型和外部库

* **编程语言：C++，HLSL Shader Model 5.0**
* **图形API：DirectX11**
* **图形界面库：ImGui图形界面库**
* **模型加载库：Assimp模型加载库**
* **项目框架：基于X\_Jun教程中的框架**
* **集成开发环境：Visual Studio 2022**

# 三．功能模块详细设计

## 地形生成模块

#### 方块设计

#### Block方块类中包含了方块的枚举Id以区分不同的方块种类；方块的数据使用了框架中的GameObject类，来存放方块的纹理、变换、网格等数据；设计了BlockModel类来专门从文件中获取方块的纹理，设置纹理模型和材质。Block方块类中还设计了相应的方法来获取和设置类中成员；

#### 区块设计

#### Chunk区块类中存放了区块的坐标；实例方块，所有的实例对象都是基于对应的实例方块绘制的；实例数据（InstancedData），用来存储每个实例对象的世界矩阵和对应的逆矩阵的转置，由此便可将实例对象放置在世界坐标中的对应位置；若开启了视锥体裁剪，还会将每个实例对象的碰撞箱变换到视锥体的局部坐标中，来判断该物体是否在可见范围内，进行进一步的筛选，但由于项目中方块的数量过于庞大，一个区块中就有近上万个方块，判断每个方块的碰撞箱是否和视锥体碰撞箱相交会给CPU带来巨大的负担，所以我设置了静态变量来控制所有区块的视锥体剔除；存放物块Id的三维数组，用于碰撞测试，使玩家不穿墙；世界种子，用于控制柏林噪声的生成。

#### 获取柏林噪声的方法将三个不同频率的柏林噪声进行叠加，返回合并后的噪声，该方法设为静态方法，可方便之后在地表生成随机物体；方块Id函数在不同的位置会返回不同的Id，在加载区块时会把每个方块的坐标传入以确定其类型；在绘制场景时会调用实例化绘制来绘制区块。

#### 

#### 地表物体生成

#### 这里使用了与区块生成相同的技术，就不过多赘述，创建物体时使用了柏林噪声和我设置的参数来确定地表位置。

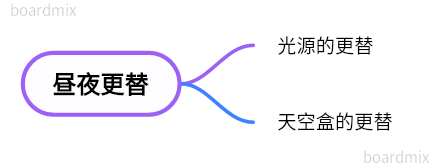
## 3.2昼夜更替模块

1**.光源的更替**

**这里我设置了四个方向光作为光源，并且将它们的漫反射系数设为可变参数，随着游戏时间的变化而减小或增大，以模拟日光的照射。**

2**.天空盒的更替**

**由于游戏开始时默认白日，所以使用了白天的天空盒，但天空盒并不受光源的影响，所以需要额外在天空盒的像素着色器中为采样的结果乘上一个系数，使着色器输出的像素相比原纹理像素更加暗淡，该系数由C++端输入并随游戏时间的变化而改变。**

**最后再调整一下两者变化的速度便可实现昼夜更替的效果。**

## 3.3特效模块

1**.雾效**

**雾效可以让远处的物体更加平滑的出现，且若是后面有时间实现区块的卸载和重**

**新加载，雾效可以让区块的加载显得不那么突兀，雾效的实现只需要设置雾效的颜色，并使用一个系数对雾效颜色和物体颜色进行线性插值即可，该系数由物体到摄像机的距离、雾效起点、雾效终点决定，也就是如下函数：**

s=(dist(**position**,**Eyes**)−fogStart​)/fogRange

**由此距离远的物体便会逐渐被雾效颜色替代。**

2**.全屏特效**

**玩游戏的时候看到在开始或退出游戏、或切换场景时会有淡入淡出的效果，这便**

**是使用了全屏特效，很多游戏也会自带小地图，这些效果都由Render-To-Texture技术实现，也就是不将后备缓冲区作为渲染目标，而是渲染到自己创建的2D纹理中，在之后的绘制中在选择性的将该纹理输出到屏幕，该技术也可以直接将渲染的效果进行输出，达到截屏的效果。**

2.1**淡入淡出**

**裁剪出一个矩形区域覆盖整个NDC空间，在该空间中为每个像素乘上一个在零到一之间变化的系数即可，该系数由C++端控制，开始游戏时由零到一，退出时由一到零。**

2.2**小地图**

**小地图只需要绘制一次即可，在游戏初始化时将相机位置设置在地形的上方，向下俯览获取全景图。之后每次绘制完场景后将小地图绘制在指定位置即可，不过要注意需要将雾效等特效关闭，不然会把雾效也绘制出来。**

## 3.4实体控制模块

## 玩家和敌人类都继承自实体类，实体拥有自身的模型和变换数据，还有实体自身的速度。实体的模型都使用了Assimp模型加载库进行导入，导入时我也遇到了点麻烦。

1.**玩家**

**玩家的控制与摄像机的控制绑定在一起，于下文详细介绍。**

2.**敌人**

**由于敌人要追踪玩家，所以需要设置一个3D向量来确定玩家与敌人的相对位置；敌人还有自身的血量，当玩家从鼠标发出的射线与敌人的模型相交时，对应的血量值减少；血量还有对应的血条，敌人血量不同时血条会切换对应的模型，来显示剩余血量，当血量为0是停止绘制敌人和血条，同时血条要时刻朝向玩家，只需对血条模型进行相应的变换即可，但也要保证血条时刻在敌人头顶，这里本来想使用公告板技术来达到这个效果，但没有成功。**

**敌人类的主要方法是寻找玩家的方法，该方法会计算出每一帧由敌人指向玩家的向量，再将敌人往该方向移动，同时血条也进行类似操作。**

# 

## 3.5摄像机控制模块

5.1**自由视角控制**

**自由视角这里沿用了框架中的自由视角控制，也就是通过键盘往四周移动，和鼠标右键旋转视野。**

5.2**第一人称控制**

**由于第一人称不能随意移动，所以这里使用了先前区块中加载好的3D数组，通过3D数组来判断人物所在位置是否有方块，来限制人物移动；还判断人物是否在地面上，来控制人物的下坠和起跳。这里将玩家的模型时刻设置在相机位置，且Y轴的旋转量与相机始终保持一致，来达到控制模型的目的**

5.3**第三人称控制**

**第三人称的人物控制与第一人称相同，只不过摄像机的准星始终朝向模型的头顶。**

## 3.6放置与破坏模块

6.1**放置方块**

**放置方块使用了碰撞箱检测，在按下按键和鼠标后会在对应位置生成一个方块模型，将该方块与周围的方块进行碰撞箱检测，若不相交则在绘制方块，否则取消放置，清除模型。**

6.2**破坏方块**

**破坏方块使用了射线检测，在鼠标处发出射线，并设置射线的范围，若与射线相交则销毁方块模型。**

## 3.7ImGui模块

6.1**控制游戏参数**

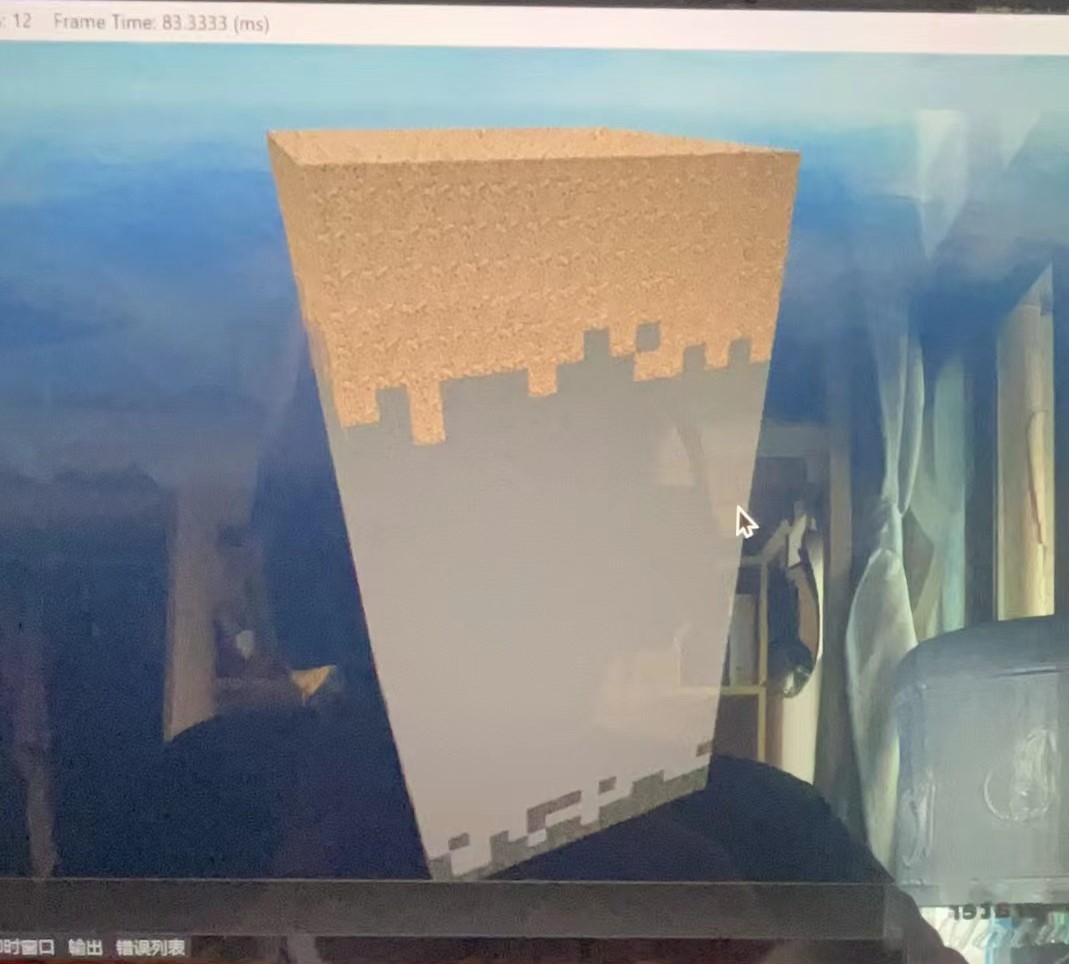
**玩家可通过ImGui控制游戏中的相机模式切换、玩家速度、第三人称观察距离等参数。**

6.2**控制特效选择**

**区块和地表物体的视锥体剔除可以使用ImGui选择性开启，雾效也可由玩家选择性开启，雾效的范围可以实时改变。**

# 四．程序演示

# 基础地形生成

**初次生成地形：**

**这便是未使用柏林噪声和实例化绘制生成的地形，可以看到仅仅生成16\*16\*32的区块帧数就掉到了12帧，这是因为所有物体的顶点都是作为顶点元素输入的，造成大量的输入装配操作，严重影响性能**

# 随后我的收到了学长给的建议，使用了硬件实例化技术进行绘制： 可以明显的看到游戏性能的到了巨大的提升，帧数直接涨到了两千多。 紧接着我又尝试使用视锥体剔除： 但帧数反而大幅下降，理由前面以进行阐明，这里不再赘述。 最后使用柏林噪声生成的地形如下图：

# 人物模型导入 导入模型并不是很顺利，一开始导入的时候人物模型的时候不知道什么原因人物的纹理并没有正常导入，导致整个模型都是白板。 随后看了学长推荐的文章后才知道OBJ模型文件都有对应的MTL文件，OBJ文件用来存储顶点索引等几何信息，MTL文件用来存储光照、透明度、折射等信息，包括使用的纹理是哪个，我使用的模型的MTL文件就是没有存放使用纹理的文件名。 最后更改后问题解决，效果如下： 相机控制 实现第一人称下模型与相机进行绑定花了挺多时间第三人称则沿用第一人称的移动方式，第三人称效果图如下：

# 添加雾效 夜晚和白天的效果如下：

# 小地图和随机生成树 效果图如下： 添加敌人

# 敌人的模型也从外部导入，也是遇到一样的问题不过事先就解决了。下面是演示敌人追踪和敌人与血条一直朝向玩家，当血量为零时敌人会被消除： 放置与破坏方块

# 这一方面我认为还不是很完善，因为我是想模拟我的世界放置方块时方块时附着在另一个方块上，效果如下：

# 五．程序亮点

## 性能优化

#### 数据库连接池

在 dao 层编写数据库连接池提高访问数据库的性能，与每次都关闭连接相比，数据库连接池在性能的优化上十分明显尤其对于即时通讯软件，对实时访问的能力要求更高，数据库连接池的作用更为明显

#### 对实时聊天用户数据使用缓存

在与 websocket 建立连接时会 将用户数据加载到内容，包括用户信息和用户所在聊天的所有成员，避免了在每次发送消息时再去查询数据库，相比之下对用户数据进行缓存的功能能够较为明显地提高用户体验

#### 使用消息缓存队列

连接时启动消息缓存队列，将消息先缓存到队列再定时缓存到数据库，对于即时通讯软件而言，频繁地发送消息，使用缓存机制能够明显降低服务器资源的消耗。

#### 线程池

在系统中创建一个线程池，每个客户端连接时会从中获取一个线程用于启动消息队列

#### 优化网页加载

优先加载消耗性能少的网页，是网页加载速度大大提升

## 用户体验

1 **提供友好的界面**

用户使用界面友好，各个地方具有使用上的提示信息

#### 简化用户操作

各个输入框组，都设置相应的键盘监听，让用户可以使用回车代替点击发送消息等操作

#### 图片上传预加载

用户上传图片时，会立即加载图片，再传回后台

1. **前端使用** ajax **进行预加载**

在前端界面上尽可能提供本地应用程序的体验，在首次加载时请求用户的聊天数据和消息记录，用户在打开新的聊天窗口时不需要重新加载，关闭聊天窗口时消息也不会被清空，包括输入框中已经输入的内容。

#### 自动加载已读消息

当请求到没有未读消息时，会自动请求每个聊天最近的 20 条已读聊天记录

#### 重复登陆提醒

当用户在别处登陆导致当前 session 失效时，会收到断开连接的提示，并且在发消息时也会收到无连接的提示

## 安全性

* + 1. 登陆过滤器：登陆校验，游客权限
    2. 后台数据检查
    3. 聊天记录独立
    4. 密码使用md5 加密,修改登陆密码.
    5. 过滤标签注入
    6. 过滤脚本攻击
    7. 防止 sql 注入

# 六．程序难点

## 前端独立窗口设计难点

**难点：聊天窗口互相覆盖，输入框，发送按钮 id 重复**

**解决：将聊天窗口和数据库表的聊天 id 一一对应，形成唯一窗口，并使用 css+javascript 实现窗口切换**

## 后台数据库表设计难点

**难点：前端需求的数据难以用一张表的数据提供，容易产生数据冗余**

**解决：构造视图层 VO 对象，对接前端需求，并且使用关联查询对接后台数据库，既实现向前端提供数据，又无需后台数据库做出妥协。**

## 后台数据库表设计难点

**难点：聊天消息只存一份，但是每个用户都可以进行管理，要求互不影响**

**解决：使用 Record 记录表作为中间表，连接用户和消息，实现聊天记录独立管理**

## 后台服务器设计难点

**难点：实时聊天要求对消息及时转发，要求尽可能降低性能开销**

**解决：查询用户数据的操作，使用用户数据缓存，保存消息记录的操作，使用定时消息缓存队列，实现聊天过程无需数据库操作**

## 后台服务器设计难点

**难点：加好友和加入聊天群组希望实现实时通知，并且无需重载用户数据缓存**

**解决：聊天服务器对外提供动态地指定哦那过户发送消息通知，动态添加聊天成员地接口，实现用户缓存热更新。**

# 七．体会总结

## 学习与收获

#### 三月初

**实践**:

刚开始学 Java，尝试过几行代码加深理解

**学习**:

刚开学，只会 C 语言，也只有寒假 C 语言课程设计项目的经验，其他的编程语言都没有学过，开学后知道工作室考核要用 java，赶紧开始学 Java。

#### 三月中旬

**实践**:

学了一些 Java 基础语法，只写过几行简单的代码，没有 Java 项目经验

**学习**:

工作室一面招新，Java 学到了接口和抽象类

#### 三月底

**实践**:

3 月 21 号开始做第一个Java 项目

**学习**:

工作室一面之后开始学数据库和 git,二面时掌握基本的 mysql 数据库操作，git 的基本操作，学了 jsp 和

servlet,以及 tomcat,并且慢慢学会使用基本的 html 标签。

#### 四月初

**实践**:

**第一个** Java **项目已完成，总结项目经验：**

1. 了解到有el 表达式和 jstl 标签库可以代替 jsp 中的 java 代码
2. 有正则表达式可以高效地进行字符匹配和数据检查。
3. 没有异常处理，只是简单打印堆栈
4. 常量没有管理，随处编写

**学习**:

在经验总结基础上，学习 Java 的 io 流，泛型，反射的知识，以及 rbac 权限管理，思考了异常的处理方案，单例模式，静态类和面向对象编程的关系，线程安全问题的出现和处理，几乎保持每天思考一个问题的节奏，在训练营中，了解到 postman 测试工具和 chrome 的调试工具，知道如何对前后端分别测试，了解到 md5 加密和 uuid。在导师的提示下，学习枚举类进行常量管理 。

#### 四月中旬

**实践**:

4 月 7 号开始做第二个 Java 项目，这次真正使用了 git 进行版本管理，并且尝试给项目添加阿帕奇开源许可证，给项目生成 javadoc 文档，在学习了反射知识的基础之上，对项目的 dao 层进行了大幅度的改进，不再每个 dao 层实现类去取连接，执行，处理结果集，而是封装在一个基础 dao 实现类统一处理，用反射实现结果集的映射。并且尝试写了一个简单的数据库连接池。

**学习**:

学习了数据库的条件查询，模糊查询，分页查询，数据库的事务隔离级别，正则表达式，Java 的内存分区，反射和类加载器，了解 gc，过滤器，监听器，session 和 cookie。

#### 四月底

**实践**:

第二个 Java 项目完成，租了一个阿里云服务器，买了个域名，尝试在远程 linux 系统上部署了自己的项目，在此过程中踩了不少坑，但是也对主机，域名，端口等概念有了更深的理解，并且在配置 tomcat 的过

程中，也对 tomcat 的目录结构更加了解，通过大量阅读 tomcat 的日志文件，处理错误，也体会到日志文件

的好处。

###### 总结项目经验:

1. 控制层没有对请求做处理，直接下放到逻辑层，导致逻辑层接口不明确，与控制层严重耦合，不利于代码维护和复用。
2. 没有业务逻辑设计，流程简单，大部分只在做增删查改，代码的价值不高。
3. 没有采用后台数据检查，依靠前端阻止不合理数据，不可靠。这个在部署项目之后发布到朋友圈，被各种注入攻击的过程中深有体会。
4. 没有采用后台数据校验，相信前端传回的登录用户名等信息，部署项目之后这个漏洞被利用，导致数据被篡改。
5. 没有限制用户的操作次数合理范围，部署项目之后发现一个用户不断发送内容相同的评论，导致服务器资源被浪费，破坏评论的有效性。

**学习**:

训练营的培训中，了解到 javascript,css,json,ajax 等一些知识，小组培训中了解到了可以使用动态代理和自定义注解实现更好的 dao 层，因此花几天学习了动态代理，注解的知识，凭借当时小组培训时的一点印象，使用动态代理，注解，工厂方法，实现了 dao 层无实现类，只有接口，让动态代理去解析注解和执行。使得 dao 层代码大大简化，效率大大提高。借此也改进了数据库连接池，使用代理连接对象，使得在关闭连接时自动将连接放回连接池。了解数据库的内连接，联表查询，联表删除。

#### 五月初

**实践**:

进入考核阶段，因为在做小组作业的过程中不断实践和积累，写 Java 代码的能力一直慢慢提升，不断地踩坑和摸索，发现和解决问题，经验也越来越多，此时 Java 已经具备了一定的基础，代码上的实现很容易就有思路，这个时候考虑更多的就是架构的设计上，业务逻辑设计和业务流程设计上，异常处理上，代码规范上，性能优化和用户体验上的一些问题。技术上也开始尝试前后端分离，除了登录注册之外，全部在前端使用 ajax 和 json 实现前后端数据的交互，并且尝试进行前端架构的设计，用 js 实现了前端页面展示的逻辑。开发的过程中使用 postman 对后台数据的接口进行测试，真正体会到前后端分离带来的好处，并且从前端，后端，两个角度理解了整个 web 程序的构建过程，和前后端协调交互，分工协作的过程。

**学习**:

在项目的技术需求下，学习了websocket技术,了解ajax轮询，长连接等知识。

## 项目体会

#### 技术上的体会:

* + 1. 出错处理很重要，我们的程序不可能一直在理想的状态下运行，不合理的数据，意外的环境变化等都有可能影响程序的执行，如果没有对数据做检查，没有做好异常处理的话，
    2. 架构设计很重要，一开始就要做到需求分析和架构设计，架构设计的合理的话可以实现各个模块充分解耦，很容易扩展，到后面在架构之上增加功能的话不容易产生耦合，并且能够复用代码，就很快
    3. 编码规范很重要，一个是代码注释，注释写好到后面很容易维护，一个是范式，比如数据库建表范式，遵循范式可以让设计更合理，少走弯路，一个是接口规范，前后端交互的时候按照接口来写，效率很高

#### 生活上的体会:

1. 第一个是对自己要有信心

刚开始拿到考核题的时候觉得很难，很多东西没有学过，很多需求也没有做过，觉得是不可能完成的，但是坚持下来，也发现还是可以做完。

1. 第二个是做事要细心

很多时候错误只要细心一点就可以避免

1. 第三个是要有恒心

从训练营开始到现在，遇到许多困难，学习的知识非常多，有些问题是一遍又一遍的测试的过程中才发现和解决的。还需要解决很多的技术难题，有恒心，坚持，很重要。

## 总结

这个学期比较忙，学的东西非常多，生活上的空闲时间很少，大部分的时间，一直在不停地思考所学的新知识，也不停地反思之前所写的项目，存在什么问题。但是偶尔还是要抽空去打打篮球，如果连续两周没有运动，加上大量时间在学习，或者遇到难题迟迟不能解决，往往会觉得身心俱疲，这个时候就会放松一下，下午的时间拿去打一下篮球，打球的过程中专注于运动，也就忘记了学习的压力，困难，烦恼，运动之后一身大汗淋漓，身体的疲劳代替了心灵的烦恼，反而倍感轻松，第二天就重新充满了活力，也更加镇定，很多问题也迎刃而解。尽管生活过得忙碌，偶尔也会在吃饭的时候给妈妈发个微信，发一发饭菜的照片，说几句家常。真的体会到了平时真的很忙也会熬夜，却跟家里人说一切都好的滋味。训练营的经

历使我认识了许多优秀的人，也领悟到许多道理。每周的培训都能学到很多的知识，更重要的是，我发现

QG 的师兄非常的负责，每周在全体培训之后，还有大组培训，这个时候后台组能学到前端的知识，前端组的能学到后台的知识，培训之后还有答疑，非常负责地解决我们的问题。小组培训还会讲很多后台实用的知识，培训之后导师还了解我们的学习情况和解决遇到的问题。每次陪训我总能感觉到 QG 的师兄是真正在帮助我们学到知识，并且学得更快更好，因此也都十分珍惜每次陪训的机会。平时的时间我也会和导师交流遇到的问题，这个过程也学到很多东西，少走很多弯路。

训练营的过程让我领悟到，做一件事，要么不做，要么认真负责地完成。也让我明白，交流是一种很好的学习方式，交流让知识更好地融合，交流促进对知识更深的理解，交流让灵感迸发。训练营的经历将成为不可磨灭的一段记忆。