MindMeet设计模型

1. 概述

1.1. 设计进展

在上一次的分析模型中，我们着重分析了`MindMeet`的系统架构以及用例类图和序列图，在这一次的设计模型中，我们将着重根据具体的平台框架，设计每个子系统和功能的具体实现。截止到目前，我们已经完成的工作有：

1. `MindMeet`的用例分析
2. `MindMeet`功能设计
3. `MindMeet`的系统架构划分
4. `MindMeet`的子系统划分
5. `MindMeet`子系统接口设计
6. `MindMeet`平台框架技术栈选择
7. `MindMeet`数据持久化，好友私聊设计机制实现
8. `MindMeet`日程规划，专注模式用例具体实现
9. `MindMeet`UI原型设计

以下是涉及到的专业术语

术语	解释
MindMeet	本软件的名称，意为促进思想交流和分享的时间管理平台
SRS	软件需求规格说明书（Software Requirements Specification）
UI	用户界面（User Interface）
API	应用程序接口（Application Programming Interface）
日程表	用来记录用户日程的对象
交流圈	用户可以在其中分享专注记录与日程表，同时对他人的分享进行评论
自习室	用户可以创建、加入的虚拟自习空间，用于自习和共同学习
专注力	集中注意力于特定任务或活动，而不受干扰或分心
专注模式	用户使用软件进行专注的软件状态
任务管理	通过记录、安排和跟踪任务来有效管理时间和工作
日程安排	通过创建和管理日程表来安排和规划时间
记录跟踪	记录专注记录用于评估时间使用情况和提高生产力

术语	解释
悄悄话	好友间私聊，交流学习
计划借鉴	指用户将导入他人分享的日程表导入自己的日程，一般用于用户像有相关经验用户借鉴学习日程安排

1.2. 平台框架

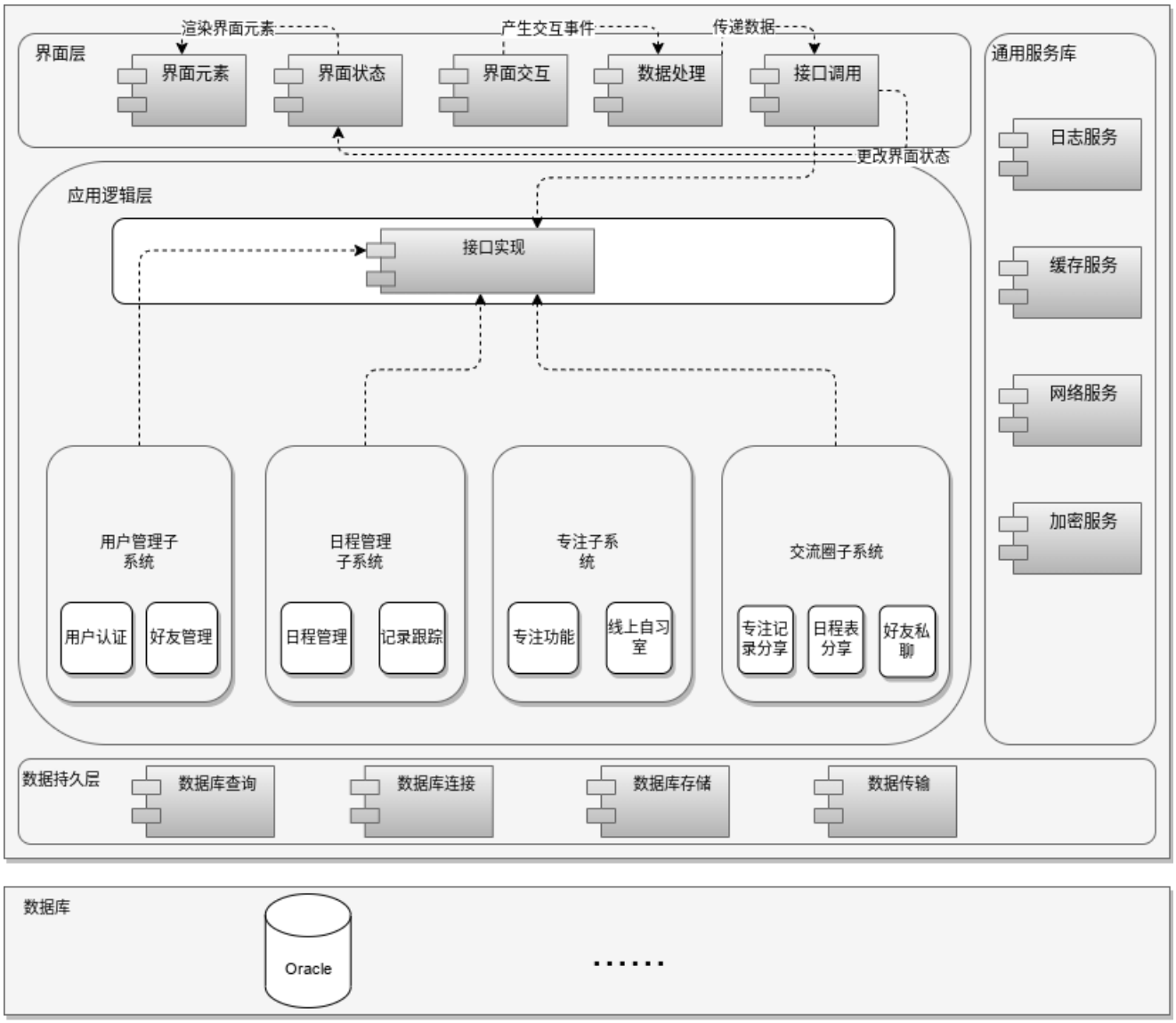
我们的项目使用分层框架，界面，业务逻辑，数据处理进行分层处理，整体依托于安卓平台与java语言开发，涉及到的相关技术栈列出如下：

- 安卓界面开发:使用 `Android Studio` , `Android SDK` , `XML` 开发与定义用户界面与布局
- 服务器框架：使用 `REST API` 来与服务器进行通信,在 `Spring Boot` 应用程序中构建RESTful Web服务，然后在安卓应用程序中使用HTTP协议来访问这些服务。
- 数据存储：我们使用Mybatis实现数据持久化，查询数据库，Oracle数据库来存储数据，使用Redis来满足高并发情况下数据的安全性和一致性。
- 日志信息：我们使用 `Log4j` 来记录日志信息，方便我们进行调试与错误处理。 `Log4j` 是Apache的一个开源日志框架，提供了丰富的日志级别、日志输出目的地、日志格式等配置选项，可以灵活地控制日志输出。
- 数据可视化：使用开源框架 `Apache ECharts` 来实现统计数据的可视化。

2. 架构完善

2.1. 平台依赖系统架构

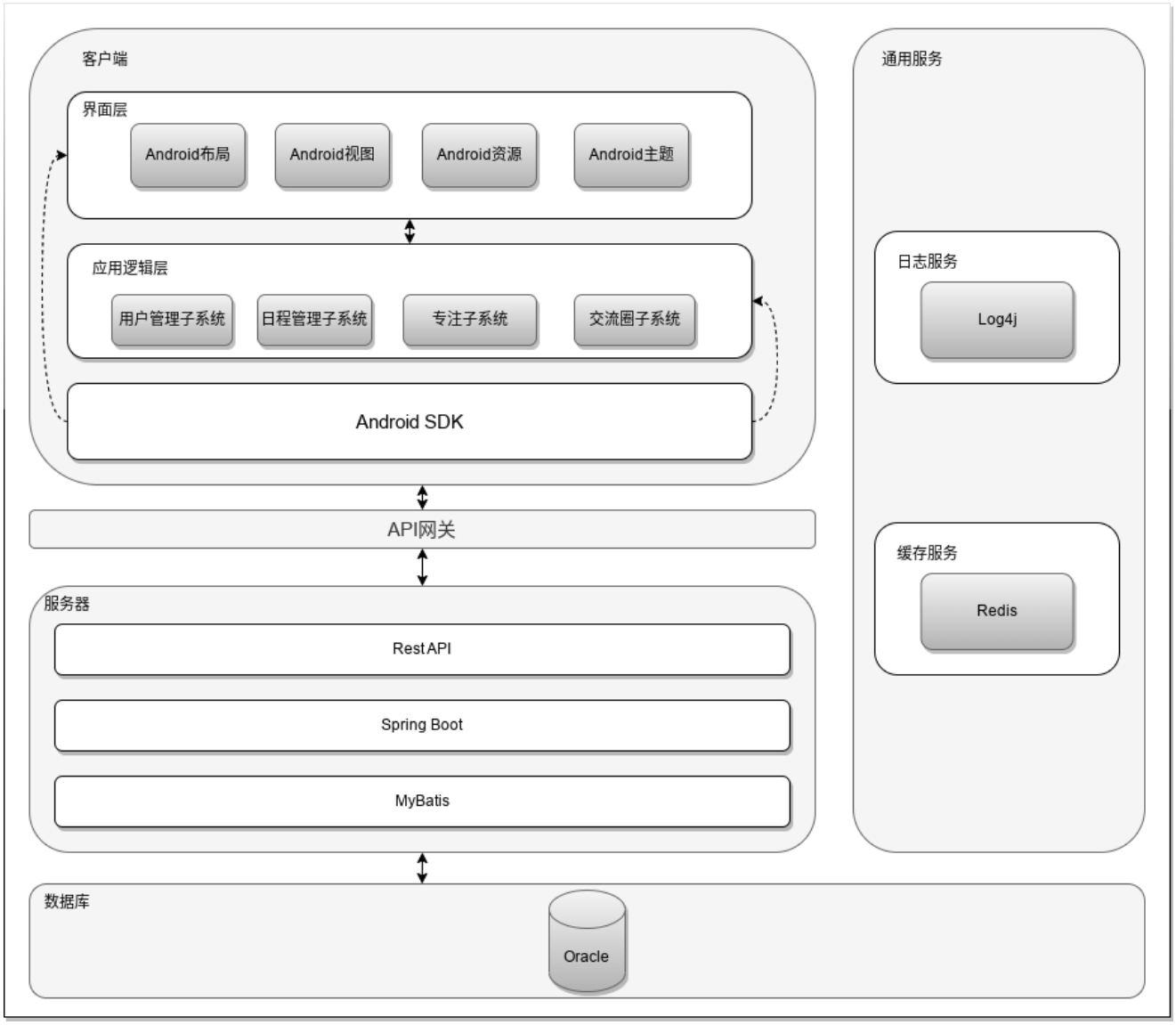
我们对上次作业的系统架构分析进行了完善，重新对系统架构进行了划分，以下是最新的系统架构：



我们将控制层删除，应用逻辑层的子系统实现接口提供界面层调用，从而更改界面层状态；同时，我们将原有的子系统重新划分，划分为：

- 用户管理子系统：实现用户认证，好友管理功能
- 日程管理子系统：实现日程管理，记录跟踪功能
- 专注子系统：实现专注功能与线上自习室功能
- 交流圈子系统：实现专注记录分享，日程表分享，好友私聊功能

上次作业的系统架构分析中,我们注重于系统的功能划分,这次我们将更加注重于系统有关平台的依赖架构，平台依赖的系统架构如下图所示：



我们采用C/S架构设计，客户端使用 `Android SDK` 相关api绘制界面，应用逻辑层中四个子系统，调用 `Android SDK` 相关api进行逻辑操作，比如禁用通知，音频播放等功能。

服务器端通过 `Spring Boot` 框架部署 `Rest Api` 与客户端进行通信，通过开源框架 `MyBatis` 封装 `JDBC` ，进行数据库的访问与写入。同时，通用服务层使用 `Log4j` 进行日志输出，使用 `Redis` 提供数据缓存服务，满足高并发数据库查询需求。

2.2. 子系统与接口

为了更好提供我们的业务服务，根据我们的app涉及到的方面进行领域划分，同时针对app提供的功能在进一步划分为以下子系统:

- 用户管理子系统
- 专注子系统
- 日程管理子系统
- 交流圈子系统

用户管理子系统

编号	Rest Api	接口描述
001	POST api/user/register	该接口接受用户注册信息，返回是否注册成功，以及对应的userID
002	PUT api/user/modify	该接口接受用户修改信息，返回是否修改成功
003	POST api/user/login	该接口接受用户登录信息，返回是否登录成功
004	GET api/user/?userID=id	该接口接受用户ID，返回用户的详细信息
005	DELETE api/user	该接口接受用户ID，删除该用户的信息
006	POST api/friendship/addFriend	该接口接受用户ID和好友ID，添加好友关系
007	DELETE api/friendship/removeFriend	该接口接受用户ID和好友ID，删除好友关系

专注子系统

编号	Rest Api	接口描述
008	POST api/focus/upload	该接口接受用户的一次上传专注记录，返回是否上传成功,以及专注id
009	GET api/focus/?userID=id	该接口接受用户ID，返回用户的所有专注记录
010	GET api/focus/?focusID=id	该接口接受一个专注记录的id,返回此专注记录的所有信息
011	DELETE api/focus/delete	该接口接受专注ID，删除该专注记录，返回是否成功
012	POST api/room/create	该接口接受线上自习室创建信息，返回是否创建成功，以及对应的roomID
013	PUT api/room/modify	该接口接受线上自习室修改信息，返回是否修改成功
014	GET api/room/?roomID=id	该接口接受线上自习室ID，返回线上自习室的详细信息
015	DELETE api/room/roomID	该接口接受线上自习室ID，删除该线上自习室的信息
016	POST api/room/join	该接口接受用户ID和线上自习室ID，加入线上自习室
017	DELETE api/room/leave	该接口接受用户ID和线上自习室ID，退出线上自习室

日程管理子系统

编号	Rest Api	接口描述
018	POST api/schedule/create	该接口接受用户的一次上传日程记录，返回是否上传成功
019	GET api/schedule/?sheduleID=id	该接口接受日程ID，返回该日程的详细信息
020	DELETE api/schedule/delete	该接口接受日程ID，删除该日程的详细信息，返回是否成功
021	GET api/schedule/?userID=id	该接口接受用户id,返回用户所有日程的日程
022	PUT api/schedule/modify	该接口接受用户的一次修改日程记录，返回是否修改成功

交流圈子系统

编号	Rest Api	接口描述
023	POST api/circle/show/create	该接口接受用户的一次分享，返回分享是否成功
024	GET api/circle/show/?showID=id	该接口分享id，返回该分享的详细信息
025	DELETE api/circle/show	该接口接受分享id，删除该分享的详细信息，返回是否成功
026	POST api/circle/comment/create	该接口接受用户的一次评论，返回评论是否成功
027	GET api/circle/comment/?showID	该接口分享id，返回该分享的评论信息
028	DELETE api/circle/comment	该接口接受评论id，删除该评论，返回是否成功
029	POST api/circle/like/create	该接口接受用户的一次点赞，返回点赞是否成功
030	GET api/circle/like/?showID=id	该接口接受分享id，返回该分享的点赞信息
031	DELETE api/circle/like	该接口接受点赞id，删除该点赞，返回是否成功

2.3. 接口规范

我们的App允许第三方开发者开发其个人的数据可视化工具，因此对外提供查询用户个人专注信息，专注记录,以及日程信息，提供以下接口：

编号	001
REST API	POST api/user/login
Interface	login(userName: String,passWord:String):bool
Arguement	userName: String,passWord:String
Return Value	bool
Introduction	此接口使用户于外部平台登陆，返回登陆相关信息

编号	002
REST API	GET api/schedule/?userID=id
Interface	getSchedule(userID: String):List
Arguement	userID: String
Return Value	List
Introduction	此接口通过userID查询用户日程表的列表

编号	003
REST API	GET api/focus/?userID=id
Interface	getFocus(userID: String):List
Arguement	userID: String
Return Value	List
Introduction	此接口通过userID查询用户专注记录的列表

编号	004
REST API	GET api/focus/?focusID=id
Interface	getFocus(focusID: String):bool
Arguement	focusID: String
Return Value	bool
Introduction	此接口通过focusID查询用户专注记录的详细信息

编号	005
REST API	GET api/schedule/?scheduleID=id
Interface	getSchedule(scheduleID: String):bool
Arguement	scheduleID: String
Return Value	bool
Introduction	此接口通过scheduleID查询用户日程的详细信息

以上接口所对应返回的json信息格式如下：

- 001：

```
{
    "code": 200,
    "msg": "success",
    "data": {
        "userID": "123456",
        "userName": "张三",
        "userAvatar": "http://xxx.com/xxx.jpg",
    }
}
```

- 002:

```
 {
     "code": 200,
     "msg": "success",
     "data": [
         {
             "scheduleID": "123456",
             "scheduleName": "计算机网络",
             "scheduleTime": {
                   "startTime": "2020-12-12 12:12",
                   "endTime": "2020-12-12 15:12"
               },
             "scheduleContent": "计网作业",
             "scheduleType": "学习",
             "scheduleStatus": "未完成"
         },
         {
             "scheduleID": "111132",
             "scheduleName": "计算机系统结构",
```

```
            "scheduleTime": {
                  "startTime": "2020-11-11 13:00",
                  "endTime": "2020-12-12 15:00"
               },
            "scheduleContent": "系统结构`作业",
            "scheduleType": "学习",
            "scheduleStatus": "未完成"
         }
    ]
}
```

- 003:

```
{
    "code": 200,
    "msg": "success",
    "data": [
       {
          "focusID": "123456",
          "focusTime": {
                "startTime": "2020-11-14 15:00",
                "endTime": "2020-11-14 17:00",
             },
       },
       {
          "focusID": "121388",
          "focusTime": {
                "startTime": "2020-10-22 13:00",
                "endTime": "2020-10-22 15:00",
             },
       },
    ],
}
```

- 004:

```
{
      "code": 200,
      "msg": "success",
      "focusID": "121388",
      "focusTime": {
            "startTime": "2020-10-22 13:00",
            "endTime": "2020-10-22 15:00",
            },
}
```

- 005:

```
 {
       "code": 200,
       "msg": "success",
       "scheduleID": "123456",
       "scheduleName": "计算机网络",
       "scheduleTime": {
             "startTime": "2020-12-12 12:12",
             "endTime": "2020-12-12 15:12"
          },
       "scheduleContent": "计网作业",
       "scheduleType": "学习",
       "scheduleStatus": "未完成"
 }
```

2.4. 日程管理子系统接口示例

日程管理子系统一共提供以下接口：

编号	001
REST API	POST api/schedule/create
Interface	ScheduleUtil.createSchedule(schedule: Schedule):bool
Arguement	schedule: Schedule
Return Value	bool
Introduction	此接口使用户创建日程，返回创建结果

编号	002
REST API	PUT api/schedule/modify

编号	002
Interface	ScheduleUtil.modifySchedule(oldScheduleID: String,newSchedule:Schedule):bool
Arguement	oldScheduleID: String,newSchedule:Schedule
Return	bool
Introduction	此接口使用户修改日程，返回修改结果

编号	003
REST API	DELETE api/schedule/delete
Interface	ScheduleUtil.deleteSchedule(scheduleID: String):bool
Arguement	scheduleID: String
Return	bool
Introduction	此接口使用户删除日程，返回删除结果

编号	004
REST API	GET api/schedule/?scheduleID=id
Interface	ScheduleUtil.getSchedule(scheduleID: String):Schedule
Arguement	scheduleID: String
Return	Schedule
Introduction	此接口通过scheduleID查询用户日程的详细信息

编号	005
REST API	GET api/schedule/?userID=id
Interface	ScheduleUtil.getScheduleList(userID: String):List
Arguement	userID: String
Return Value	List
Introduction	此接口通过userID查询用户日程列表

编号	006
Interface	ScheduleUtil.getScheduleListByTime(userID: String,startTime: String,endTime: String):List
Arguement	userID: String,startTime: String,endTime: String
Return Value	List

编号	006
Introduction	此接口通过userID和时间段查询用户日程列表

编号	007
Interface	ScheduleUtil.getScheduleListByType(userID: String,scheduleType: String):List
Arguement	userID: String,scheduleType: String
Return Value	List
Introduction	此接口通过userID和日程类型查询用户日程列表

编号	008
Interface	ScheduleUtil.getScheduleListByStatus(userID: String,scheduleStatus: String):List
Arguement	userID: String,scheduleStatus: String
Return Value	List
Introduction	此接口通过userID和日程状态查询用户日程列表

添加日程操作流程

1. 通过前端界面获取到用户输入的日程信息
2. 构造Schedule对象
3. 调用 `ScheduleUtil.createSchedule(schedule: Schedule):bool` 接口
4. 得到调用结果，返回给前端

- 传递json格式示例：

```
{
    "code": 200,
    "msg": "success"
}
```

修改日程操作流程

1. 通过前端界面获取到用户输入的修改后日程信息和要修改的日程ID
2. 构造Schedule对象
3. 调用 `ScheduleUtil.modifySchedule(oldScheduleID: String,newSchedule:Schedule):bool` 接口
4. 得到调用结果，返回给前端

- 传递json格式示例：

```
{
   "code": 200,
   "msg": "success"
}
```

删除日程操作流程

1. 通过前端界面获取到要删除的日程ID
2. 调用 `ScheduleUtil.deleteSchedule(scheduleID: String):bool` 接口
3. 得到调用结果，返回给前端

- 传递json格式示例：

```
{
   "code": 200,
   "msg": "success"
}
```

查询当前用户日程列表操作流程

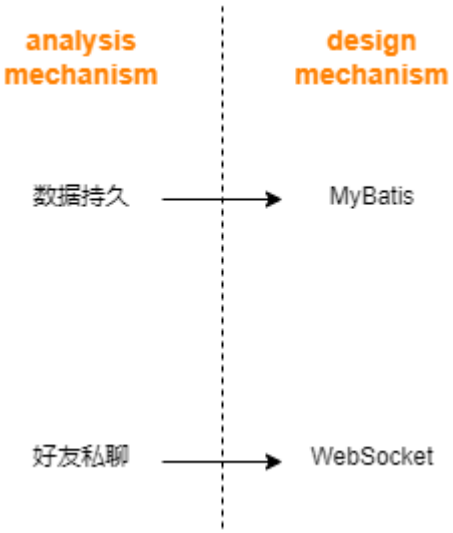
1. 用户登陆后获取到userID
2. 调用 `ScheduleUtil.getScheduleList(userID: String):List<Schedule>` 接口
3. 得到调用结果，返回给前端绘制日程列表

- 传递json格式示例：

```
  {
     "code": 200,
     "msg": "success",
      "data": [
          {
              "scheduleID": "123456",
              "scheduleName": "计算机网络",
              "scheduleTime": {
                    "startTime": "2020-12-12 12:12",
                    "endTime": "2020-12-12 15:12"
                },
              "scheduleContent": "计网作业",
              "scheduleType": "学习",
              "scheduleStatus": "未完成"
```

```
            },
            {
                "scheduleID": "111132",
                "scheduleName": "计算机系统结构",
                "scheduleTime": {
                      "startTime": "2020-11-11 13:00",
                      "endTime": "2020-12-12 15:00"
                   },
                "scheduleContent": "系统结构作业",
                "scheduleType": "学习",
                "scheduleStatus": "未完成"
            }
        ]
    }
```


3. 设计机制



3.1. 数据持久机制

分析机制

我们的MindMeet具有好友私聊和日程分享功能，随着时间的推移和用户量的增长，系统将持有大量的数据信息，如好友的聊天记录、用户的日程记录。通过将数据存储在共享的持久介质中，多个应用程序或用户可以共享和访问相同的数据。这种共享和协作可以实现数据的一致性和互操作性，促进不同系统之间的数据交换和集成。

什么是数据持久层呢？数据持久层的主要目的是实现数据的长期存储和管理，以确保数据在应用程序关闭、系统重启或断电等情况下仍然可用。它允许应用程序将数据从内存中持久化到物理存储介质中，以便在需要时进行读取和修改。数据持久层通常与数据库系统密切相关，它可以使用各种数据库管理系统（DBMS）来存储和检索数据，例如关系型数据库（如MySQL、Oracle）、面向对象数据库（如MongoDB）或内存数据库（如Redis）等。它还可以与文件系统、消息队列和其他持久

化机制集成，以满足不同类型的应用程序需求。在数据持久层中，通常使用各种技术和方法来管理数据的存储和检索，例如数据访问对象（DAO）模式、对象关系映射（ORM）工具、查询语言（如SQL）和索引技术等。这些技术和方法可以简化数据操作的过程，并提供高效的数据管理和访问能力。

在此我们使用数据持久机制作为MindMeet的数据存储和访问，这样做的好处有：

1. 利于数据长期存储

   据持久机制确保数据在应用程序关闭、系统重启或断电等情况下仍然可用。将数据存储到持久介质中，如数据库或文件系统，可以避免数据丢失或不可恢复的情况。

2. 实现数据的一致性和互操作性，促进不同系统之间的数据交换和集成

   通过将数据存储在共享的持久介质中，多个应用程序或用户可以共享和访问相同的数据。

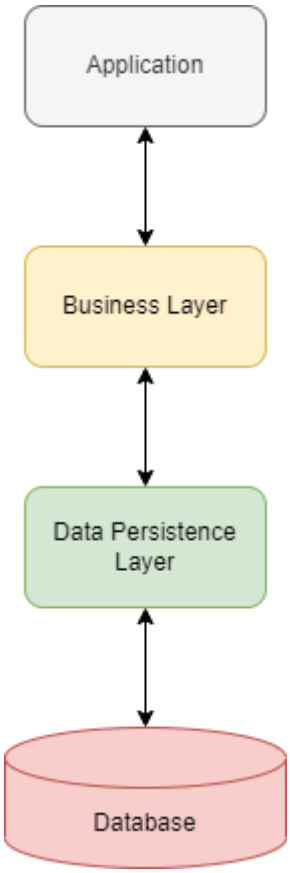
3. 拥有数据完整性和一致性

   通过使用事务和数据约束等机制，数据持久机制可以确保数据的完整性和一致性。它可以实施数据验证规则、强制执行数据完整性规则，并提供事务处理来保证数据的正确性和一致性。

4. 数据具有安全性

   数据持久机制提供了各种安全措施来保护存储的数据。它可以实施访问控制、加密、备份和灾难恢复等措施，以保护数据免受未经授权的访问、数据泄露和数据损坏等风险。

数据持久层（Data Persistence Layer）是指在计算机系统中负责将数据存储到持久介质（如硬盘、数据库）中，并能够在需要时检索和更新数据的部分。它是应用程序与底层数据存储之间的接口层。在MindMeet中数据持久层在我们的业务层（里面是我们的子系统）与数据库之间。持久层充当中间介质，可以将对象存储在数据库中，也可以将数据对象加载到内存中。在这样的抽象层作用下，程序只会间接访问数据库。



实现机制

现在有了数据持久层的分析机制(analysis mechanism)，于此将采用MyBatis做为其的实现机制(design mechanism)。

MyBatis基本概念

1. MyBatis是一个半ORM（对象关系映射）框架，它内部封装了JDBC，开发时只需要关注SQL语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement等繁杂的过程。程序员直接编写原生态sql，可以严格控制sql执行性能，灵活度高。
2. MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO映射成数据库中的记录，避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。
3. 通过xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java对象和 statement 中sql的动态参数进行映射生成最终执行的sql语句，最后由mybatis框架执行sql并将结果映射为java对象并返回。（从执行sql到返回result的过程）。

MyBatis的底层操作封装了JDBC的API，MyBatis的工作原理以及核心流程与JDBC的使用步骤一脉相承，MyBatis的核心对象（SqlSession，Executor）与JDBC的核心对象（Connection，Statement）相互对应。

MyBatis的工作原理

我们知道，JDBC有四个核心对象：

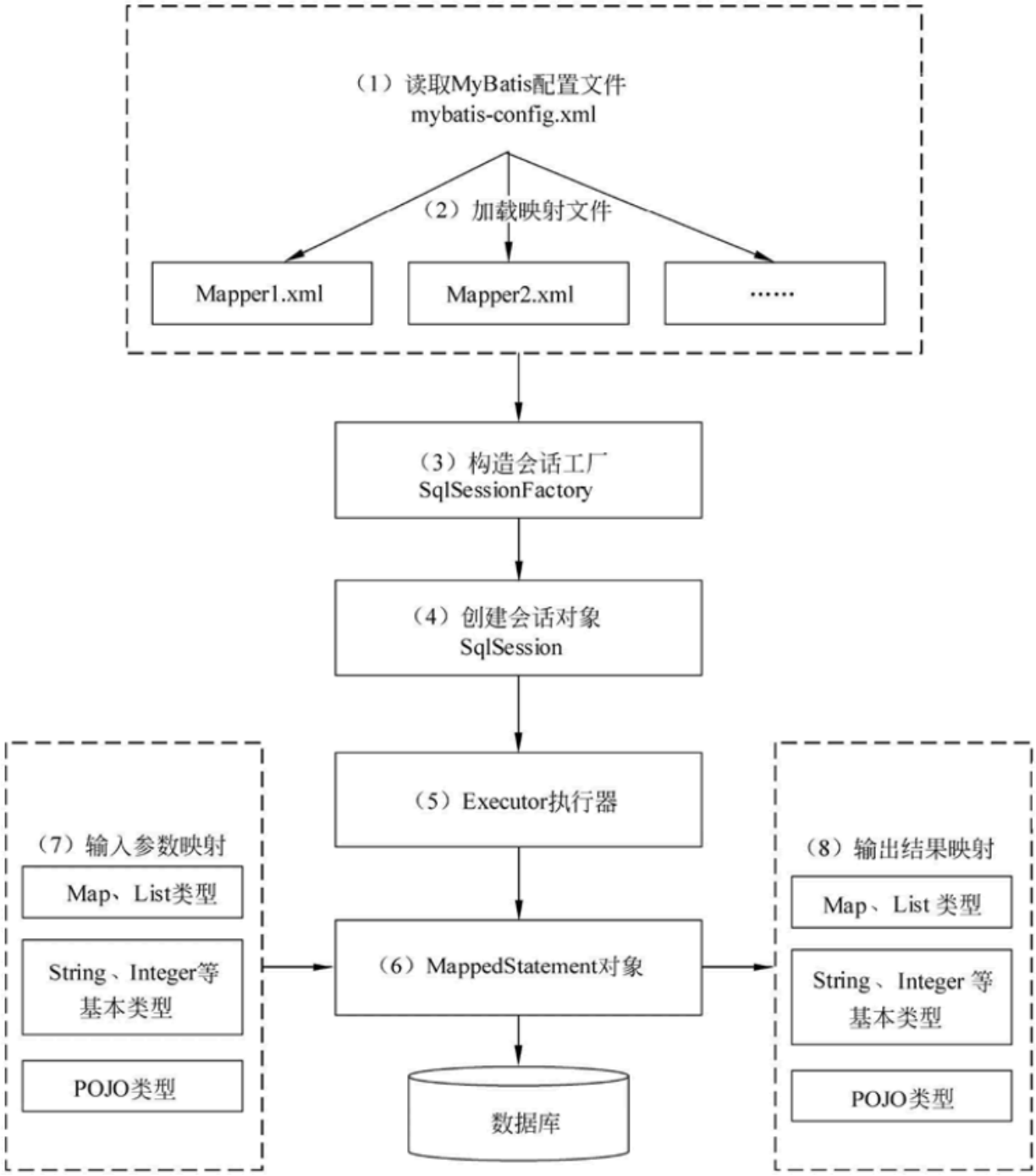
1. **DriverManager**，用于注册数据库连接
2. **Connection**，与数据库连接对象

3. **Statement/PrepareStatement**，操作数据库SQL语句的对象
4. **ResultSet**，结果集或一张虚拟表

而MyBatis也有四大核心对象：

1. **SqlSession**对象，该对象中包含了执行SQL语句的所有方法。类似于JDBC里面的Connection 。
2. **Executor**接口，它将根据SqlSession传递的参数动态地生成需要执行的SQL语句，同时负责查询缓存的维护。类似于JDBC里面的Statement/PrepareStatement。
3. **MappedStatement**对象，该对象是对映射SQL的封装，用于存储要映射的SQL语句的id、参数等信息。
4. **ResultHandler**对象，用于对返回的结果进行处理，最终得到自己想要的数据格式或类型。可以自定义返回类型。

MyBatis的工作原理如下图所示：



上面中流程就是MyBatis内部核心流程，每一步流程的详细说明如下文所述：

1. 读取MyBatis的配置文件。mybatis-config.xml为MyBatis的全局配置文件，用于配置数据库连接信息。
2. 加载映射文件。映射文件即SQL映射文件，该文件中配置了操作数据库的SQL语句，需要在MyBatis配置文件mybatis-config.xml中加载。mybatis-config.xml 文件可以加载多个映射文件，每个文件对应数据库中的一张表。
3. 构造会话工厂。通过MyBatis的环境配置信息构建会话工厂SqlSessionFactory。
4. 创建会话对象。由会话工厂创建SqlSession对象，该对象中包含了执行SQL语句的所有方法。
5. Executor执行器。MyBatis底层定义了一个Executor接口来操作数据库，它将根据SqlSession传递的参数动态地生成需要执行的SQL语句，同时负责查询缓存的维护。

6. MappedStatement对象。在Executor接口的执行方法中有一个MappedStatement类型的参数，该参数是对映射信息的封装，用于存储要映射的SQL语句的id、参数等信息。
7. 输入参数映射。输入参数类型可以是Map、List等集合类型，也可以是基本数据类型和POJO类型。输入参数映射过程类似于JDBC对preparedStatement对象设置参数的过程。
8. 输出结果映射。输出结果类型可以是Map、List等集合类型，也可以是基本数据类型和POJO类型。输出结果映射过程类似于JDBC对结果集的解析过程。

3.2. 好友私聊机制

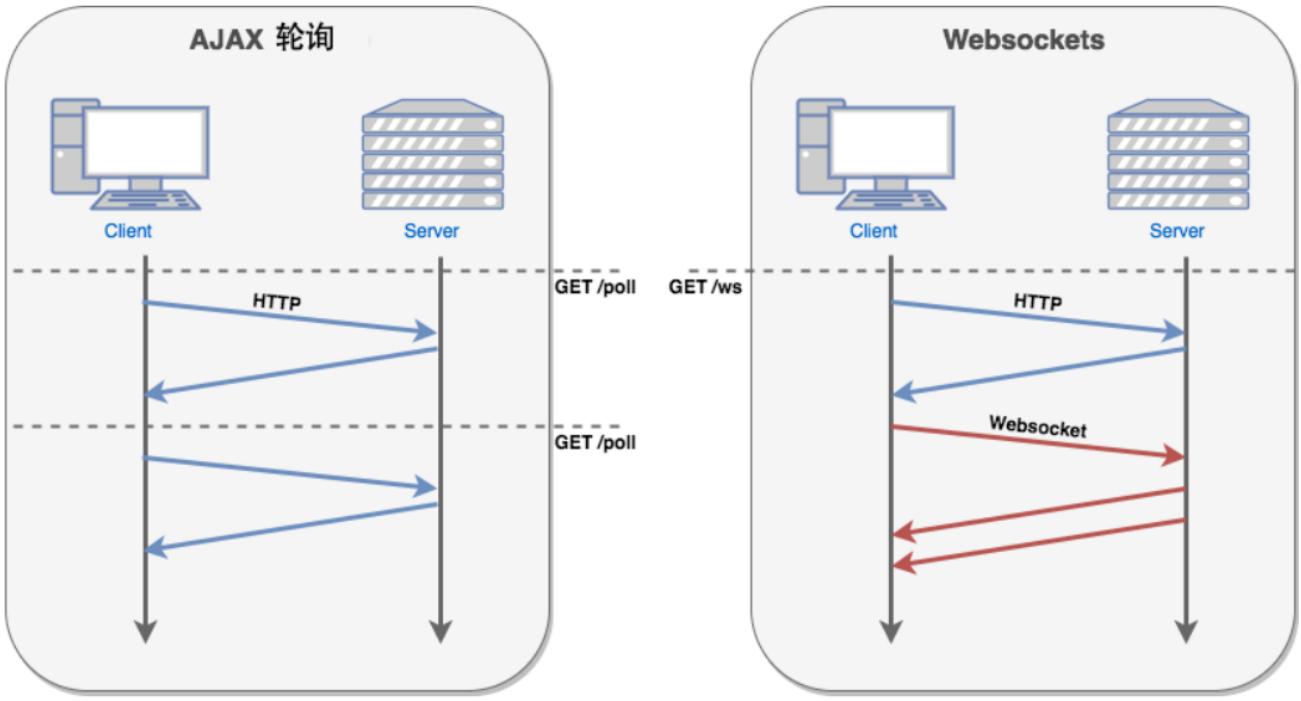
分析机制

MindMeet用于基础的信息交流功能，发送文字、表情，分享日程等。不像登录信息这种不需要很频繁或仅获取一次的数据，好友通讯需要我们通过网络传输的实时更新或连续数据流。信息传输需要具有可靠性和稳定性：信息传输应该是可靠和稳定的，确保信息能够有效地传递到目标地点。使用可靠的网络和通信设备，实施错误检测和纠正机制，以确保数据的准确性和完整性。

传统的请求-响应模式的 Web 开发在处理此类业务场景时，通常采用实时通讯方案，常见的是：

- **轮询：**
  原理简单易懂，就是客户端通过一定的时间间隔以频繁请求的方式向服务器发送请求，来保持客户端和服务器端的数据同步。问题很明显，当客户端以固定频率向服务器端发送请求时，服务器端的数据可能并没有更新，带来很多无谓请求，浪费带宽，效率低下。

传统 Web 模式在处理高并发及实时性需求的时候，会遇到难以逾越的瓶颈，我们需要一种高效节能的双向通信机制（即WebSocket机制）来保证数据的实时传输。以下是两者的实现差别：



HTTP	WEBSOCKET
单向协议	双向通信

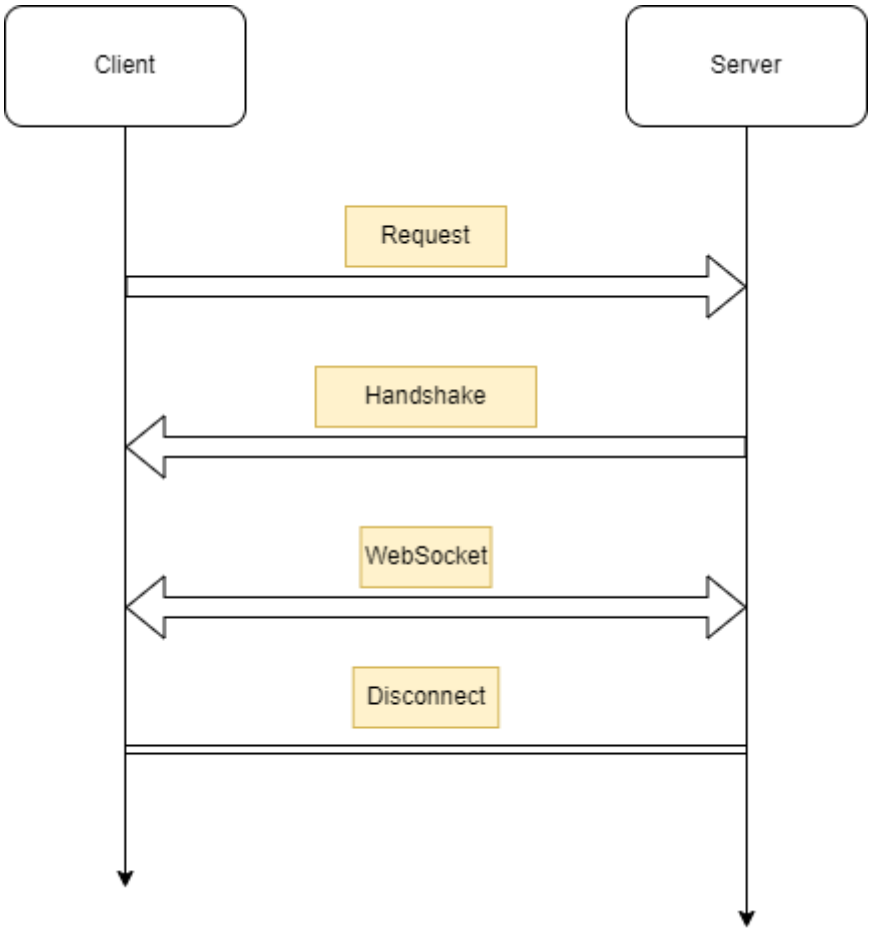
HTTP	WEBSOCKET
简单的RESTful应用	实时应用程序
只想获取一次数据供应用程序使用	经常更新的应用程序

实现机制

WebSocket基本概念

WebSocket是一种基于TCP的通信协议，它提供了全双工、持久化的连接，允许在客户端和服务器之间进行实时的双向通信。WebSocket 的实现分为客户端和服务端两部分，客户端（通常为浏览器）发出 WebSocket 连接请求，服务端响应，实现类似 TCP 握手的动作，从而在浏览器客户端和 WebSocket 服务端之间形成一条 HTTP 长连接快速通道。两者之间后续进行直接的数据互相传送，不再需要发起连接和相应。

WebSocket的基本流程



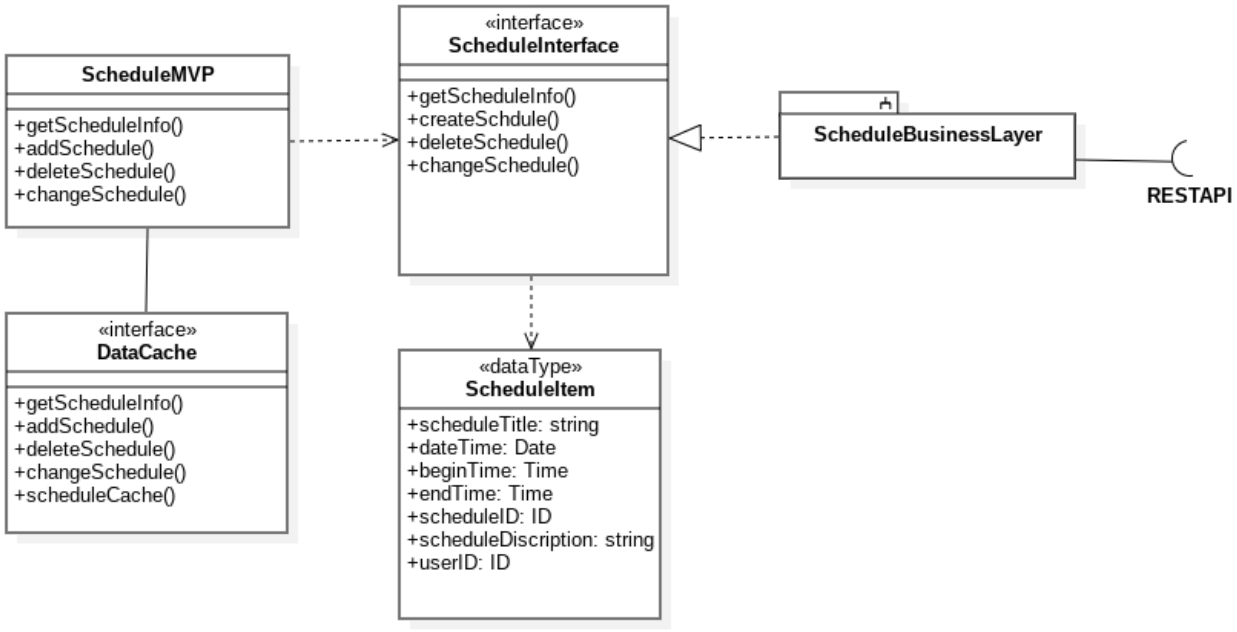
1. 建立连接：
    - 客户端通过WebSocket协议发起连接请求，请求的目标是服务器的WebSocket端点（URL）。
    - 服务器接收到连接请求后，如果同意建立连接，将返回一个HTTP状态码101（切换协议）作为响应，表示切换到WebSocket协议。
    - 客户端接收到101响应后，确认连接建立成功。

2. 双向通信：
   - 一旦WebSocket连接建立成功，客户端和服务器之间可以进行实时的双向通信。
   - 客户端和服务器可以使用WebSocket提供的API方法，如send()和onmessage()来发送和接收消息。
   - 客户端和服务器可以随时发送消息给对方，并即时接收对方发送的消息，实现实时的双向通信。
3. 保持连接：
   - WebSocket连接是持久化的，保持活动状态，直到其中一方显式关闭连接或发生通信错误。
   - 客户端和服务器可以保持连接打开，以便随时进行通信，而无需频繁地创建和销毁连接。
   - 保持连接可以减少通信延迟，并允许实时通信的持续进行。
4. 关闭连接：
   - 客户端或服务器可以选择关闭WebSocket连接。
   - 关闭连接可以通过调用WebSocket对象的close()方法来实现。
   - 关闭连接后，客户端和服务器之间的通信将终止。

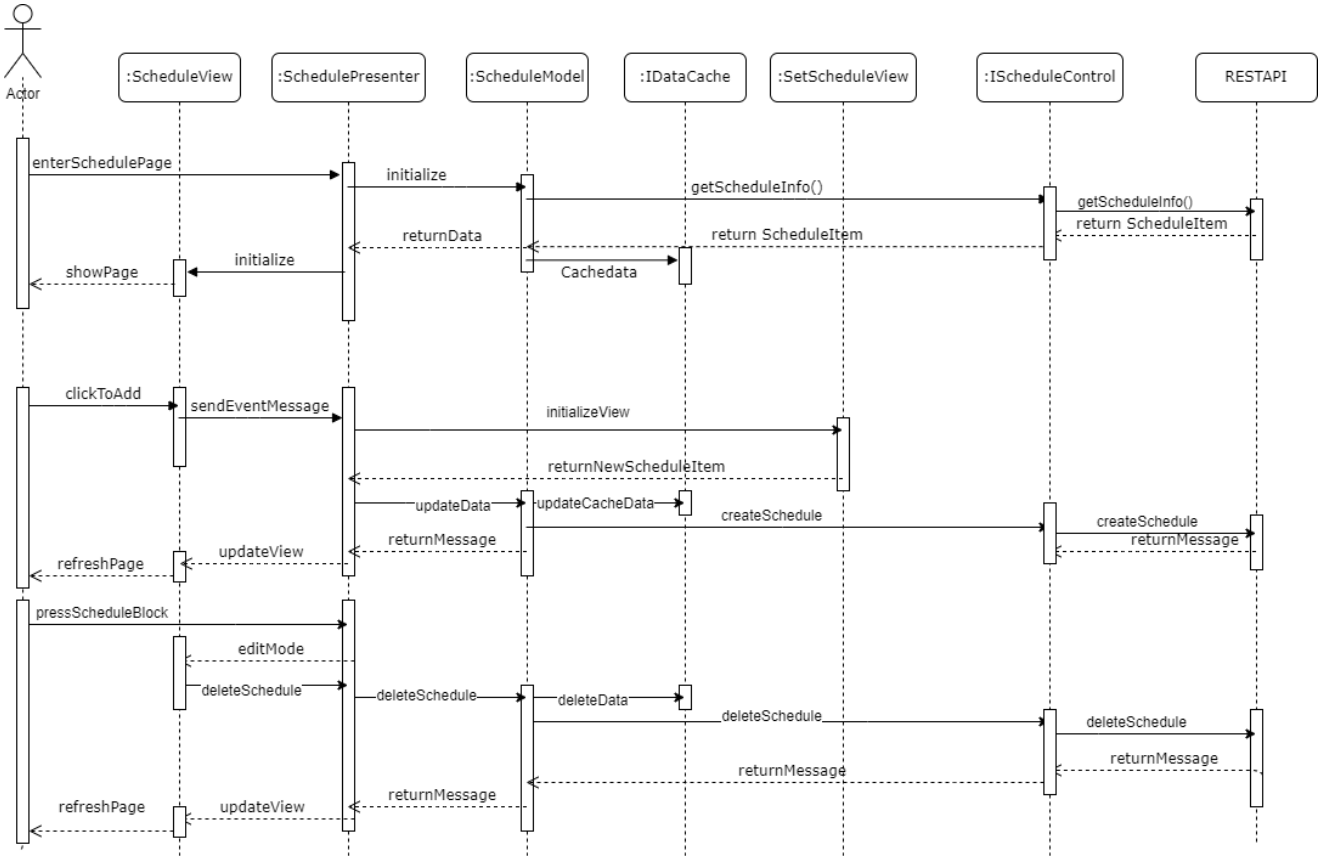
WebSocket的流程简单来说就是建立连接、进行双向通信、保持连接和关闭连接。这种实时的双向通信机制使得WebSocket在需要实时更新和实时交互的应用程序中非常有用。

4. 用例实现

4.1. 日程规划

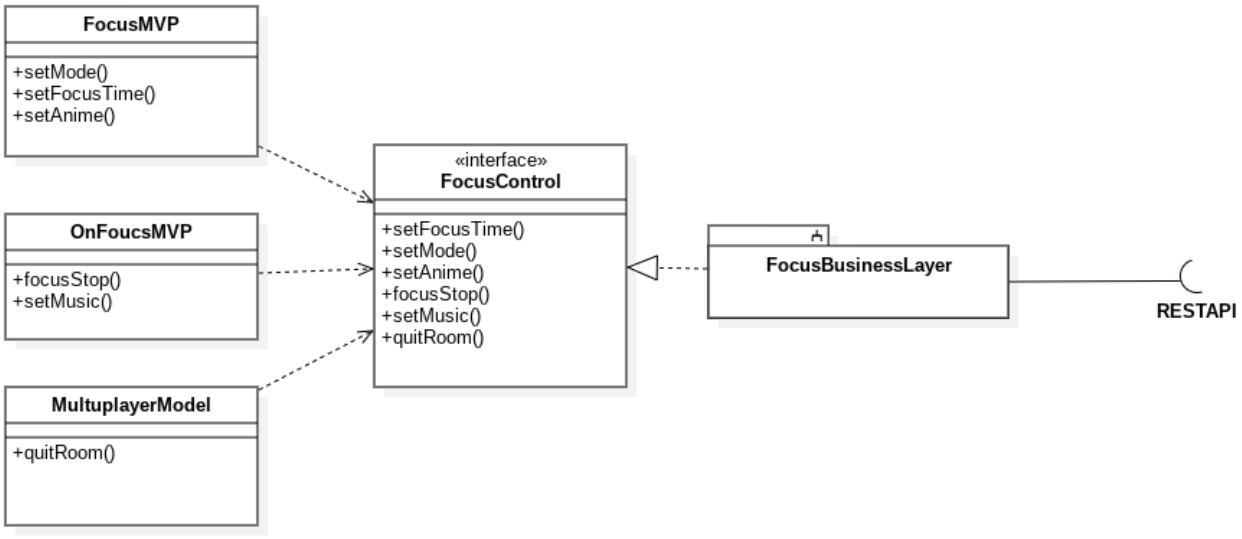


日程规划的用例实现类图中，我们按不同层级将对应用例的实现进行了划分，如上图所示，表示层主要通过MVP架构进行了实现，然后在表示层中若是需要进行对数据的增删改查等操作需要调用业务逻辑层实现的相关接口。而业务逻辑层在实现向上的接口中，再通过RESTAPI远程调用，访问服务器上的数据持久层提供的接口，进而实现对数据库的增删改查。

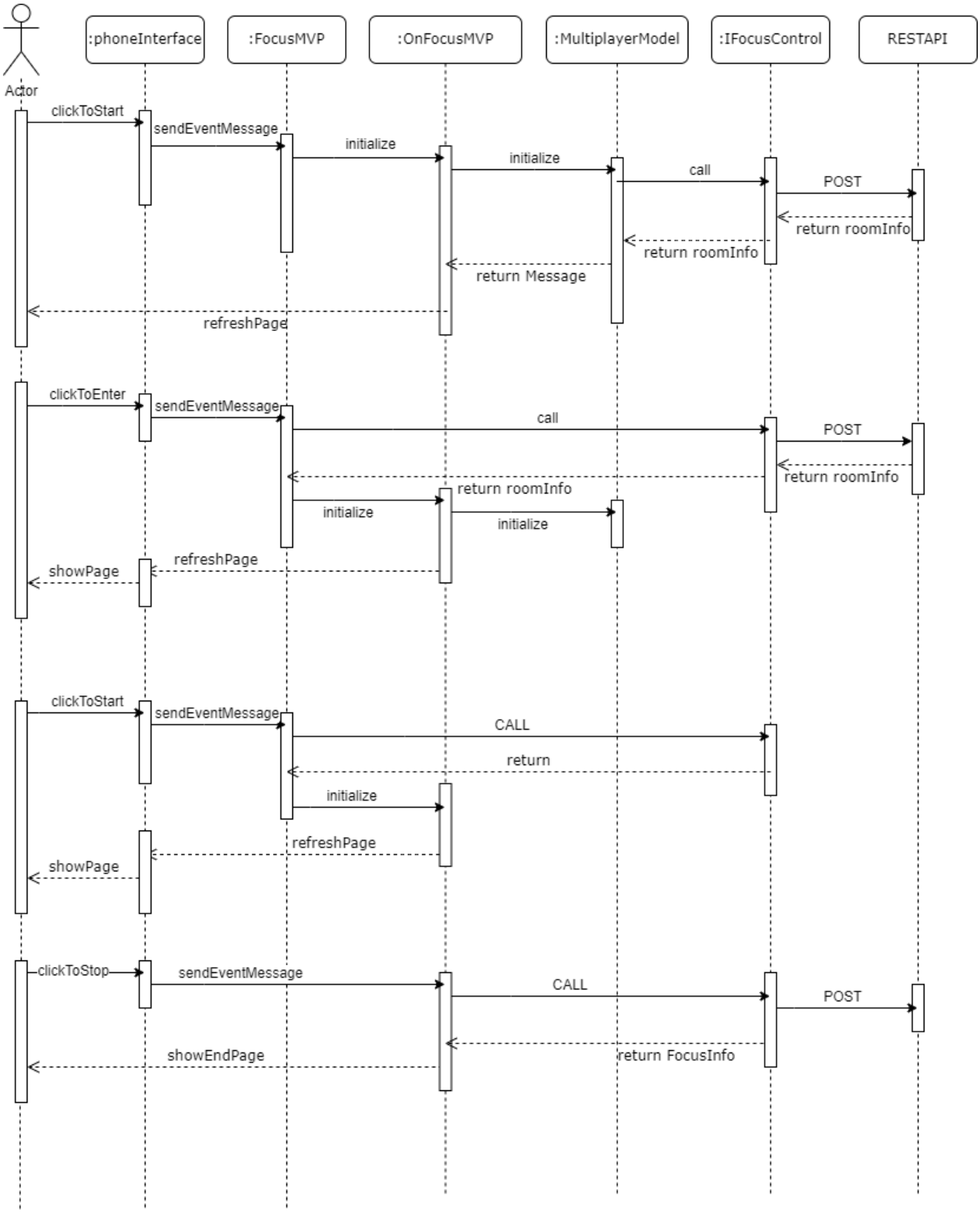


日程规划中主要的功能需求有已有日程的渲染展示，日程添加，日程修改和日程删除。上述功能实现大致流程相同，均是MVP表示层中进行相关事件的捕获，Model对象处理事件中调用业务逻辑层接口，业务逻辑层进而在接口实现中通过https的RESTAPI调用服务器的下层接口，而接口返回的信号经过业务逻辑层传输到MVP表现层进行界面的更新。这里要说明的是当本地客户端第一次向服务器申请用户日程数据时，考虑到日程相关信息基本都是文本，服务器会返回用户所有的日程数据（内存允许范围内），客户端会将其存储在本地客户端的缓存中，若用户切出软件再返回，或者选择对日程进行排序、筛选等操作时，则可以从本地缓存中迅速读取数据实现界面加载。

4.2. 专注模式



同日程规划的用例实现类图，专注模式中也通过不同层级将实现进行了划分，包括MVP表示层，业务逻辑层接口，服务端的数据持久层接口以及数据库系统。表示层通过事件驱动进行接口调用，业务逻辑层在接口实现中通过RESTAPI进行远程服务端的接口调用，最终进行数据库数据的修改和查询。



在单人专注模式中，计时会放在本地进行，只有当用户点击退出或者倒计时模式下时间结束，则表示层会处理该事件并进行下层接口调用，业务逻辑层才会通过RESTAPI向服务器端上交本次专注的相关数据，从而减少了服务器端的负担。而在多人专注模式中，由于需要与服务器进行多用户的消息同步，因此本地会额外初始化MultiplayerModel对象进行相关的表示层数据处理，建立自习室时，表示层会调用下层接口，业务逻辑层在接口中调用服务器相关接口来向服务器进行自习室相关资源的申请和建立，建立成功后消息返回到业务逻辑层进行相关处理包装后，再返回到表示层进行界面的更新，即进入自习室等待界面。进入自习室的流程也相似，用户在界面填入房间码后，表示层会将相关信息传入并调用接口，业务逻辑层中打包好信息并调用服务端接口来寻找对应的房间，

若成功找到，服务器返回成功消息与房间相关信息，业务逻辑层处理过后再返回相关信息到表示层，进入对应自习室等待界面。

5. 架构风格与关键决定

5.1 架构风格

`MindMeet` 采用分层架构，具体原因有以下几点：

1. 分层架构是一种非常常见的架构风格，它将整个应用程序划分为若干层，每一层都有自己的职责和功能，层与层之间通过接口进行通信，层与层之间的耦合度较低，易于维护和扩展。
2. `MindMeet` 不存在需要单独部署的服务，每项功能都是相互配合的，因此不需要采用微服务架构。
3. 采用分层架构，`MindMeet` 可以将前端和后端分离，前端和后端可以独立开发，前端和后端之间通过接口进行通信，前端和后端的耦合度较低，易于维护和扩展。

5.2 关键决策与设计模式

5.2.1 前端设计模式

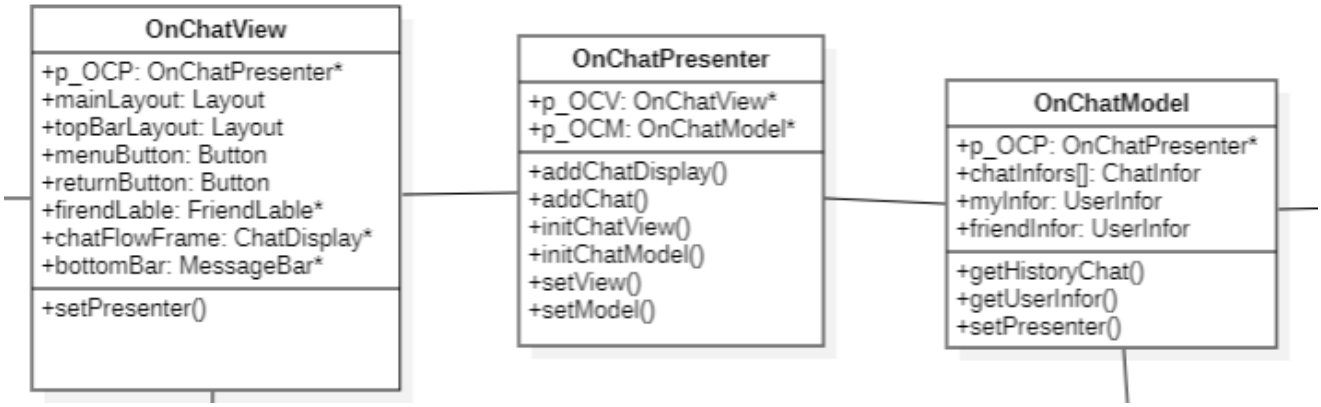
前端设计模式采用MVP模式，MVP是模型（Model）、视图（View）、主持人（Presenter）的缩写，分别代表项目中3个不同的模块。

- 模型（Model）：负责处理数据的加载或者存储，比如从网络或本地数据库获取数据等；
- 视图（View）：负责界面数据的展示，与用户进行交互；
- 主持人（Presenter）：相当于协调者，是模型与视图之间的桥梁，将模型与视图分离开来。

MVP模式的优点

1. 分离了视图逻辑和业务逻辑，降低了耦合
2. Activity只处理生命周期的任务，代码变得更加简洁
3. 视图逻辑和业务逻辑分别抽象到了View和Presenter的接口中去，提高代码的可阅读性
4. Presenter被抽象成接口，可以有多种具体的实现，所以方便进行单元测试
5. 把业务逻辑抽到Presenter中去，避免后台线程引用着Activity导致Activity的资源无法被系统回收从而引起内存泄露和OOM

MVP示例



5.5.2 后端设计模式

软件后端大部分功能均需要采用观察者模式，其用于实现对象之间的一对多依赖关系。当一个对象的状态发生变化时，该模式下可以实现自动通知并更新所有依赖对象，这些依赖对象即为观察者（Observer）

- 在交流圈中，每个客户端需要注册观察自己发布的相关分享的专注记录或日程从而能够当其他用户点赞或评论自己的分享时能够接受相关通知。同时也可以实现对用户好友相关状态的观察，当好友有新的分享或状态时，用户可以接受相关通知。
- 在线上自习室中，为了保障自习室的同步性，客户端需要注册观察服务端对应自习室内各用户的状态，因而当存在其他用户退出自习室或产生其他状态时，用户客户端能够及时接受信息并更新界面，实现线上同步。

观察者模式优点

1. 分离了主题和观察者，减少了对象之间的依赖关系，进行的解耦
2. 可以动态添加和移除观察者，在好友相关实现中方便用户对好友的删除和添加后仍能接受来自新好友的通知
3. 该模式下实现了一对多的通信机制，例如在线上自习室中，自习室内某一用户状态的变化可以同时通知给自习室内其他用户，实现多个对象之间的协作和通信。
4. 将关注点分离为主题和观察者，使得双方的职责更加清晰明确，便于代码的理解、维护和调试。

6. 原型进展

6.1 数据可视化实现

数据可视化部分使用开源框架 `Apache ECharts` 实现，由于该技术是在web平台中应用，应用到安卓平台上需要使用安卓的 `webview` 组件，通过 `Gson` 解析json文件，在后端与前端 `webview` 中传递专注数据。完整实现数据可视化功能步骤如下：

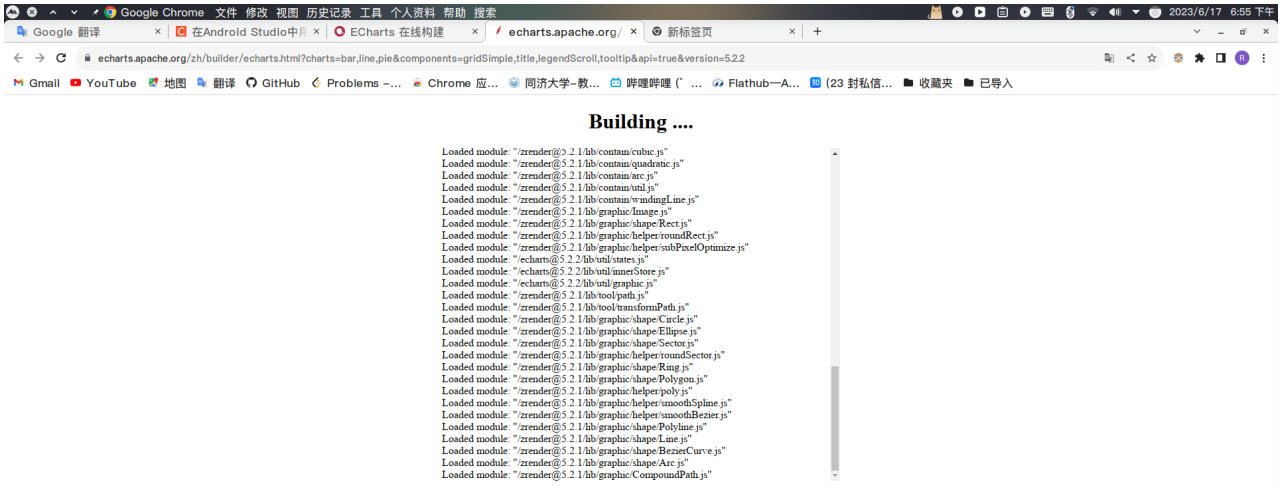
1. 添加所需依赖库

在项目 `build.gradle` 文件中添加：

```
implementation 'com.github.abel533':ECharts:3.0.0.2
implementation files('libs/gson-2.8.0.jar')
```

2. 导入 `ECharts`

到[ECharts官方网站](#)选择在线定制所需图表类型，定制完成后下载所得到 `echarts.min.js` 文件添加到项目 `app/src/main/assets/js` 文件夹中。



3. 编写 `webview` 所需 `html` 文件

在 `app/src/main/assets` 文件夹下新建 `index.html`，内容如下：

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>Document</title>
</head>

<body style="height: 100%;margin: 0">
    <h1>
        专注统计
    </h1>
    <div id="main" style="height: 300%"></div>
```

```
    <script type="text/javascript" src="./js/echarts.min.js">
</script>
    <script type="text/javascript">
        // 基于准备好的dom，初始化echarts实例
        var dom = document.getElementById("main")
        var myChart = echarts.init(dom);
        var app = {};
        function loadEcharts(echartJson) {
            var option = JSON.parse(echartJson);
            if (option && typeof option === "object") {
                myChart.setOption(option, true);
            }
        }
    </script>
</body>

</html>
```

4. 创建 `EchartsView` 用于展示数据可视化图表

```
package com.luxingzhi27.myapplication;

import android.annotation.SuppressLint;
import android.content.Context;
import android.util.AttributeSet;
import android.webkit.WebSettings;
import android.webkit.WebView;

import com.github.abel533.echarts.json.GsonOption;

public class EchartView extends WebView {
    private static final String TAG =
EchartView.class.getSimpleName();

    public EchartView(Context context) {
        this(context, null);
    }

    public EchartView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public EchartView(Context context, AttributeSet attrs, int
defStyleAttr) {
        super(context, attrs, defStyleAttr);
        init();
```

```
    }

    @SuppressLint("SetJavaScriptEnabled")
    private void init() {
        WebSettings webSettings = getSettings();
        webSettings.setJavaScriptEnabled(true);
        webSettings.setJavaScriptCanOpenWindowsAutomatically(true);
        webSettings.setSupportZoom(false);
        webSettings.setAllowContentAccess(true);
        webSettings.setAllowFileAccess(true);
        webSettings.setDisplayZoomControls(false);
        loadUrl("file:///android_asset/index.html");
    }

    /**刷新图表
     *      * java调用js的loadEcharts方法刷新echart
     *      * 不能在第一时间就用此方法来显示图表，因为第一时间html的标签还未加载完成，不能获取到标签值
     *      * @param option
     */
    public void refreshEchartsWithOption(GsonOption option) {
        if (option == null) {
            return;
        }
        String optionString = option.toString();
        String call = "javascript:loadEcharts('" + optionString + "')";
        loadUrl(call);
    }
}
```

5. 创建 `EchartOptionUtil` 处理数据json构建

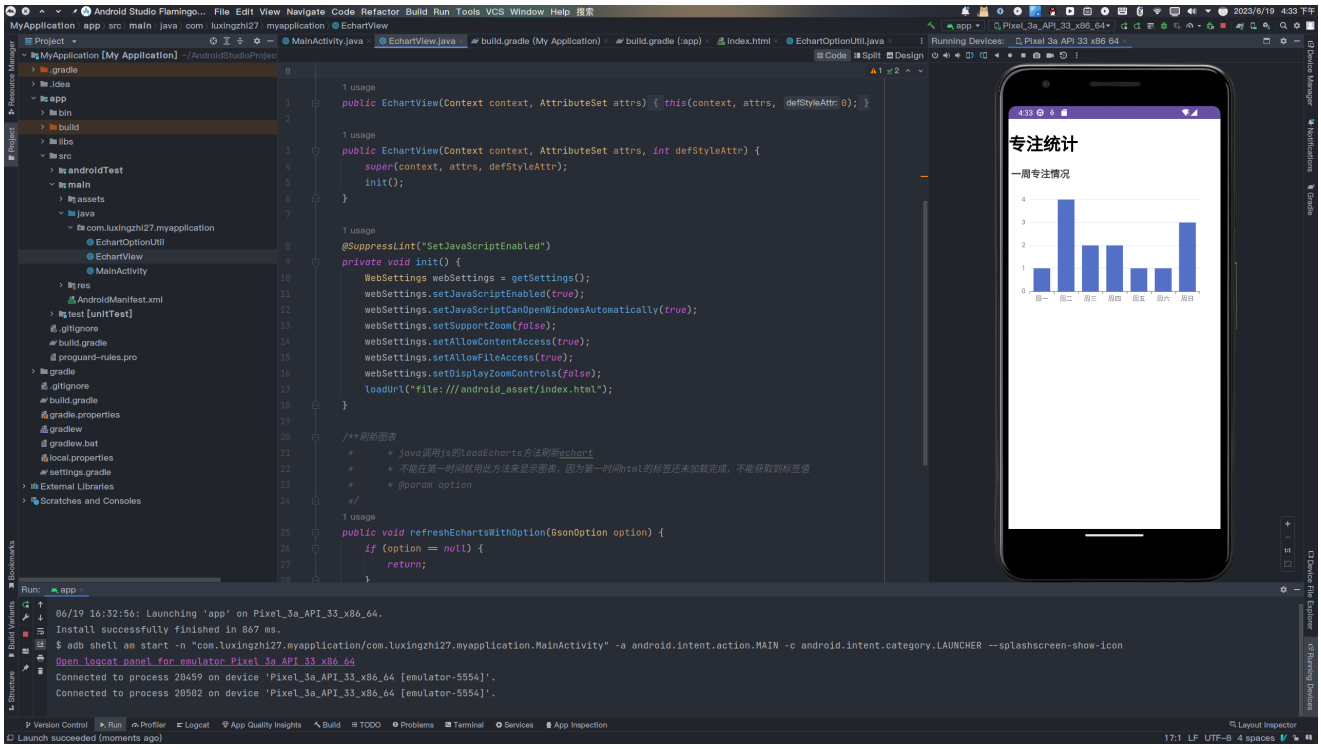
```
package com.luxingzhi27.myapplication;

import com.github.abel533.echarts.axis.Axis;
import com.github.abel533.echarts.axis.CategoryAxis;
import com.github.abel533.echarts.axis.ValueAxis;
import com.github.abel533.echarts.code.Trigger;
import com.github.abel533.echarts.json.GsonOption;
import com.github.abel533.echarts.series.Bar;
import com.github.abel533.echarts.series.Line;

import java.util.ArrayList;
import java.util.List;

```

```
public class EchartOptionUtil {

    public static GsonOption getWeekFocusLineChartOption(Object[] dayFocusTime ) {
        GsonOption option = new GsonOption();
        option.title("一周专注情况");
        option.legend("时长/小时");
        option.tooltip().trigger(Trigger.axis);

        Object[] xAxis = new Object[]{
                "周一", "周二", "周三", "周四", "周五", "周六", "周日"
        };

        ValueAxis valueAxis = new ValueAxis();
        option.yAxis(valueAxis);

        CategoryAxis categorxAxis = new CategoryAxis();
        categorxAxis.axisLine().onZero(false);
        categorxAxis.boundaryGap(true);
        categorxAxis.data(xAxis);
        option.xAxis(categorxAxis);

        Bar bar = new Bar();
        bar.name("专注时长").data(dayFocusTime).itemStyle().normal();
        option.series(bar);

        return option;
    }
}
```

6. 传入专注时长，调用刷新图表

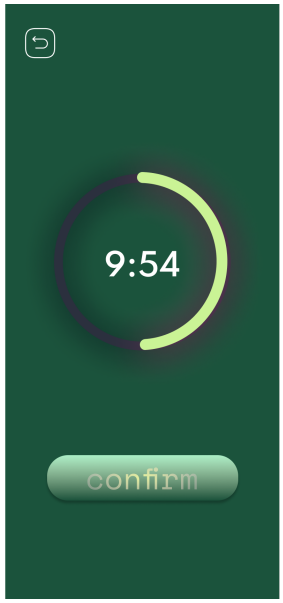
```
private void refreshLineChart(){
        Object[] dayFocusTime= new Object[]{
                (int)(Math.random()*5+1),(int)(Math.random()*5+1),(int)(Math.random()*5+1),(int)(Math.random()*5+1),
                (int)(Math.random()*5+1),(int)(Math.random()*5+1),(int)(Math.random()*5+1)
        };

 lineChart.refreshEchartsWithOption(EchartOptionUtil.getWeekFocusLineChartOption(dayFocusTime));
    }
```

显示结果：

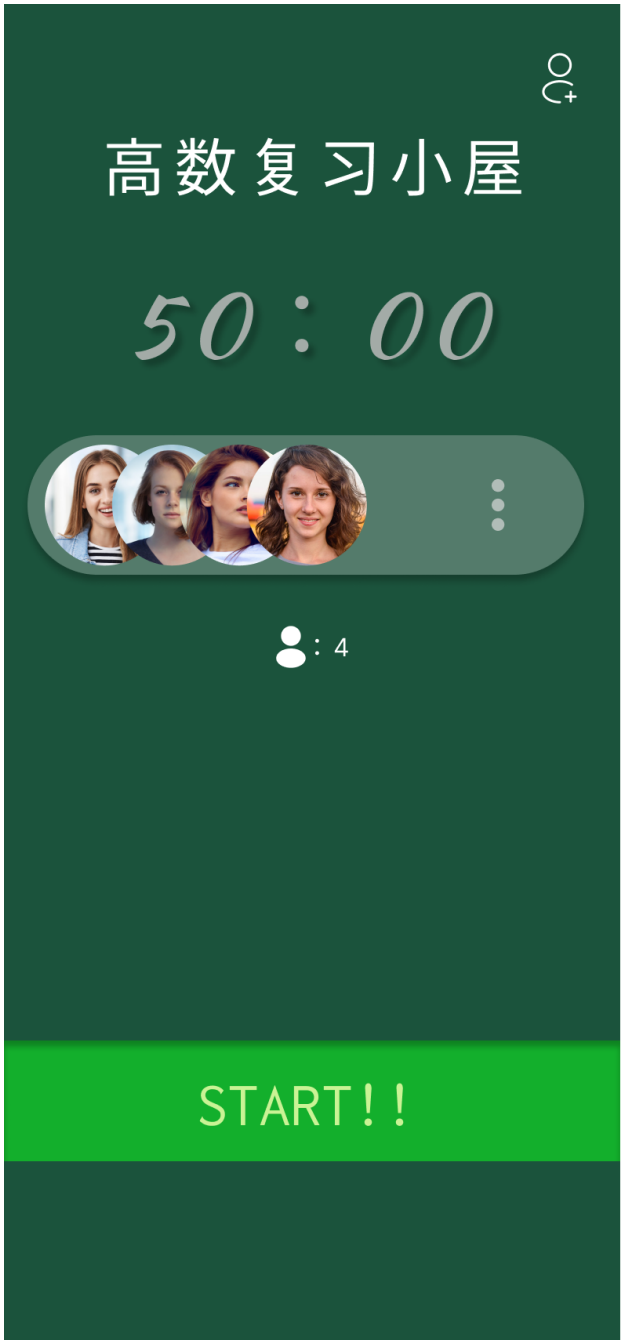


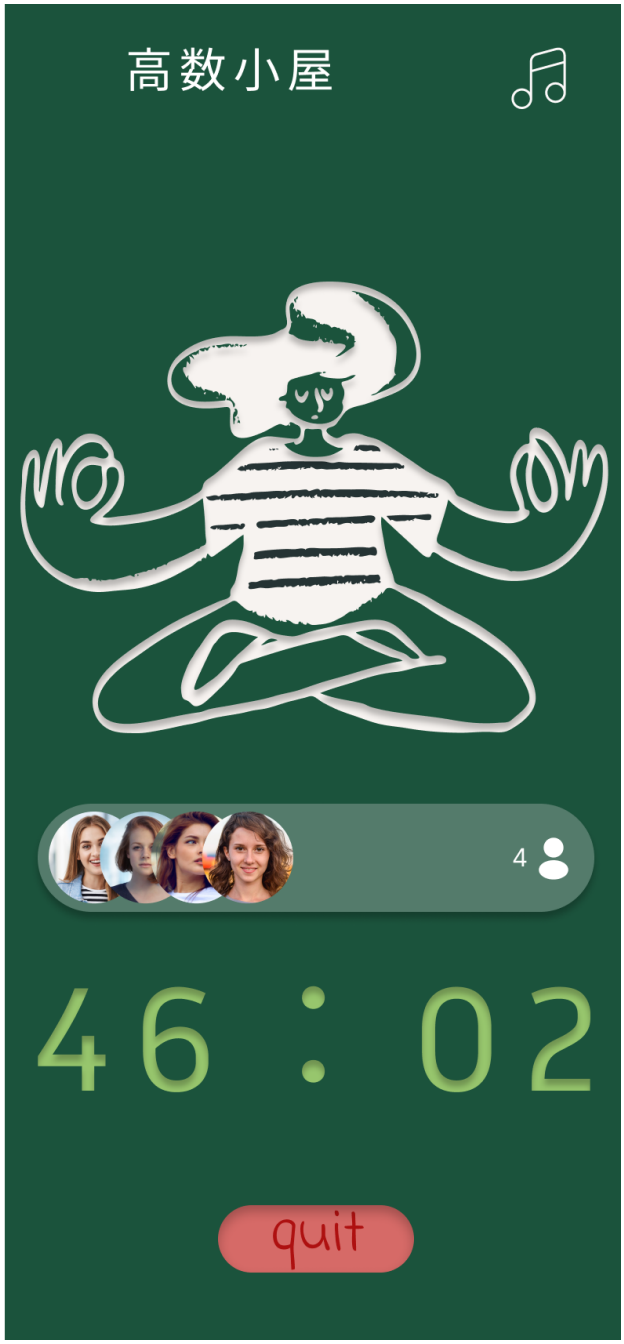
6.2 UI改进

6.2.1 专注

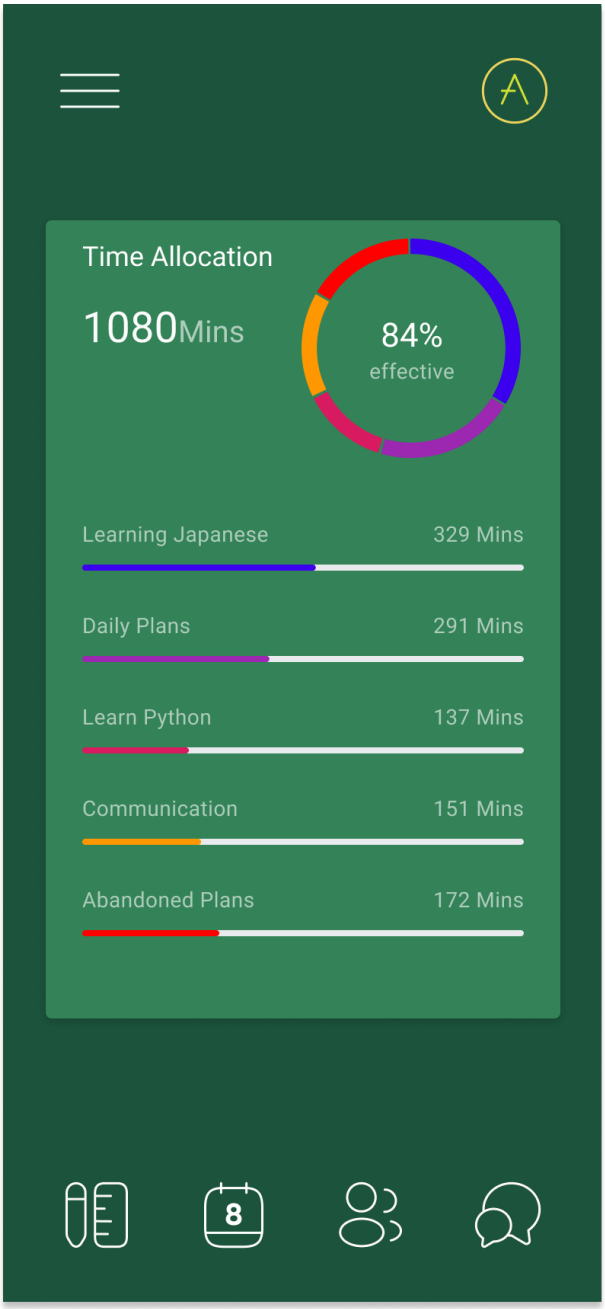


6.2.2 线上自习室





6.2.3 数据可视化



7. 存在的问题

7.1 关于社区交流，好友私聊等的管理

我们的app中涉及到一部分社交功能，这就要求我们的app要对每个人的言行举止都要有一定程度的约束与管理，目前这部分内容我们还没有进行详细的设计，有待于日后的完善。

7.2 外部系统的接入

我们希望用户能够以已有的社交账号来登陆我们的app，这就要求我们要对一些外部系统进行接入，与外部系统接入的详细流程我们也还没有详细设计考虑。

7.3 扩展功能

我们希望能够将程序的一些功能接口对外开放，供第三方开发者开发属于自己的时间管理的相关插件，这部分内容也还没有相应的详细设计，有待于日后完善。

8. 项目反思

2152057 杨瑞华

在本次项目中，我们团队开发了一个专注于时间管理的自律App,从产品需求分析到分析模型，最后到设计模型，逐步递进，让我们的一个原先只是存在于脑海中的新奇想法转化为现实。

我学习了让一个系统从想法到现实的完整设计流程，同时也学会了如何与小组成员更加高效地交流。在学习的过程中，我从原先对于程序设计的过程性思维转换到系统分析设计的更高层次的设计与架构思维，这对于我以后的学习工作甚至生活方面都大有益处，另一方面，通过对项目原型，UI的设计，我也学到了更多新的技术，比如安卓应用开发相关技术和UI设计相关工具的使用。

我们的项目还有很多的不足，我们会在以后的时间里将会持续改进，努力做到更好。

2152831 陈峥海

在本次项目中，我深刻体会到团队分工时，沟通的重要性。从理论上的摄取再到项目实际中的开发，这种实际应用的过程是新奇且艰辛的，项目构思、设计模式、架构风格、成员设计……一步一步走来，一次一次从先前的基础上完善的感觉是令人自豪，同时从后往前看也能改进当初的瑕疵。

我学会了使用UML构建流程图、ER模型、序列图等等，同时还会基本使用一些界面设计的软件，这个课程对我而言是受益匪浅的。同时设计模式、设计机制的思想提高了我对程序实现的理解，让我能从更高纬度俯瞰实现流程。

在此感谢老师指出我们项目中的不足之处。

2153691 邓岳衡

本次项目主要内容是带有线上自习室，个人专注和日程管理等带有社交属性手机软件。整个学期设计下来，在核心功能上面相比与其他系统级别的结构来说可能是相对简单的（也可能是自己没有真正见识到对应的商业解决方案），即使是如此在实际设计的时候还是感觉app整体架构设计非常困难。

一方面是没有实际代码，单纯依靠眼前的一亩三分地要去尝试看清整个软件对于没有经验的大学生来说果然还是非常困难，但也因此整个项目从空想到如今有点轮廓的过程中，还是学习到非常多的相关知识。首先就是各种设计原则，设计模式等，以及对大型软件方案制定前期流程的理解，其次就是通过项目的设计也了解了许多业界在用的技术栈，框架方案等。

9. 成员贡献

1. 2152057 杨瑞华：平台依赖框架，架构图更新，子系统接口设计，子系统接口示例，数据可视化原型设计，文档编写
2. 2152831 陈峥海：设计机制实现，UI设计，文档编写
3. 2153691 邓岳衡：详细用例实现，UI设计，文档编写