实验

2023-2024
第一学期

自动排考模块

.NET平台技术——实验七展示

黄勖 黄子安 王昭栋 邹慧

01

排考要求回顾

01 排考要求回顾

要求

考场分配表：

- 分配记录ID
- 考试ID
- 考场ID
- 监考1报名ID
- 监考2报名ID
- 是否正确分配（枚举：正确分配，分配但有问题，无法分配）
- 分配规则描述（字符串存一个JSON，表述无法分配时，是由于上述5条规则无法处理）

自动排考模块：

1. 每个考场的两个监考*必须*来自不同的部门；
2. [提高要求]每个考场的两个监考*必须不能*是两个新老师（即两个老师都没有参加过去年的监考）；
3. [提高要求]每个考场的两个监考尽可能是老带新（即一个参加过去年的监考，一个没有参加过）；
4. 每个考场的两个监考尽可能性别不同；
5. [提高]如果监考性别比例无法男女搭配，那么尽可能的让少的那种性别分开一些（比如，十个考场，只有5位男老师，那尽可能的让男老师在1,3,5,7,9号考场，而不是1,2,3,4,5号考场）；
6. 在无法分配时，必须显示考场无法分配的原因。

02

接口设计

02 接口设计

Controller层的自动排考接口SchedulingExam
输入待安排的考试id
输出每个相关考场的安排信息

```
[HttpPost("/scheduling/{id}")]
public ActionResult SchedulingExam(int id)
{
    var ret=this.schedulingService.schedulingExam(id);
    return Ok(new {
        success="success",
        msg="Ok",
        data=ret
    });
}
```

02 接口设计

Service层的自动排考业务
输入待安排的考试id
输出每个相关考场的安排信息

```
public List<ScheResult> schedulingExam(int id)
{
    List<Room> rooms = (from r in _context.Room
                        join er in _context.ExamRoom on r.id equals er.roomId
                        where er.examId == id
                        select r).ToList();
    Console.WriteLine(rooms);

    List<Teacher> teachers = _context.Teacher.Where(t => t.ExamId == id).ToList();
    Console.WriteLine(teachers);
    return schedule(id, teachers, rooms);
}
```

02 接口设计

数据库设计
仅用到相关的表和有用的字段

```
namespace ExamSchedulingSystem.db
{

    public enum Campus
    {
        思明校区,
        海韵校区,
        翔安校区
    }
    [Table("teacher")]
    public class Teacher
    {
        [Key]
        public int id { get; set; }
        public int DeptId { get; set; }//工作单位
        public int Gender { get; set; }
        public Campus ExamCampus { get; set; }//监考校区
        public bool ParticipatedLastYear { get; set; }
        public int ExamId { get; set; }
        public int getid()
        {
            return id;
        }
    }
}
```

```
namespace ExamSchedulingSystem.db
{
    [Table("room")]
    public class Room
    {
        [Key]
        public int id { get; set; }
        public string? name { get; set; }
        public Campus? campus { get; set; }//校区
        public int? buildingNo { get; set; }//楼号
        public int? roomNo { get; set; }//房间号

        public int getid()
        {
            return id;
        }
    }
}
```

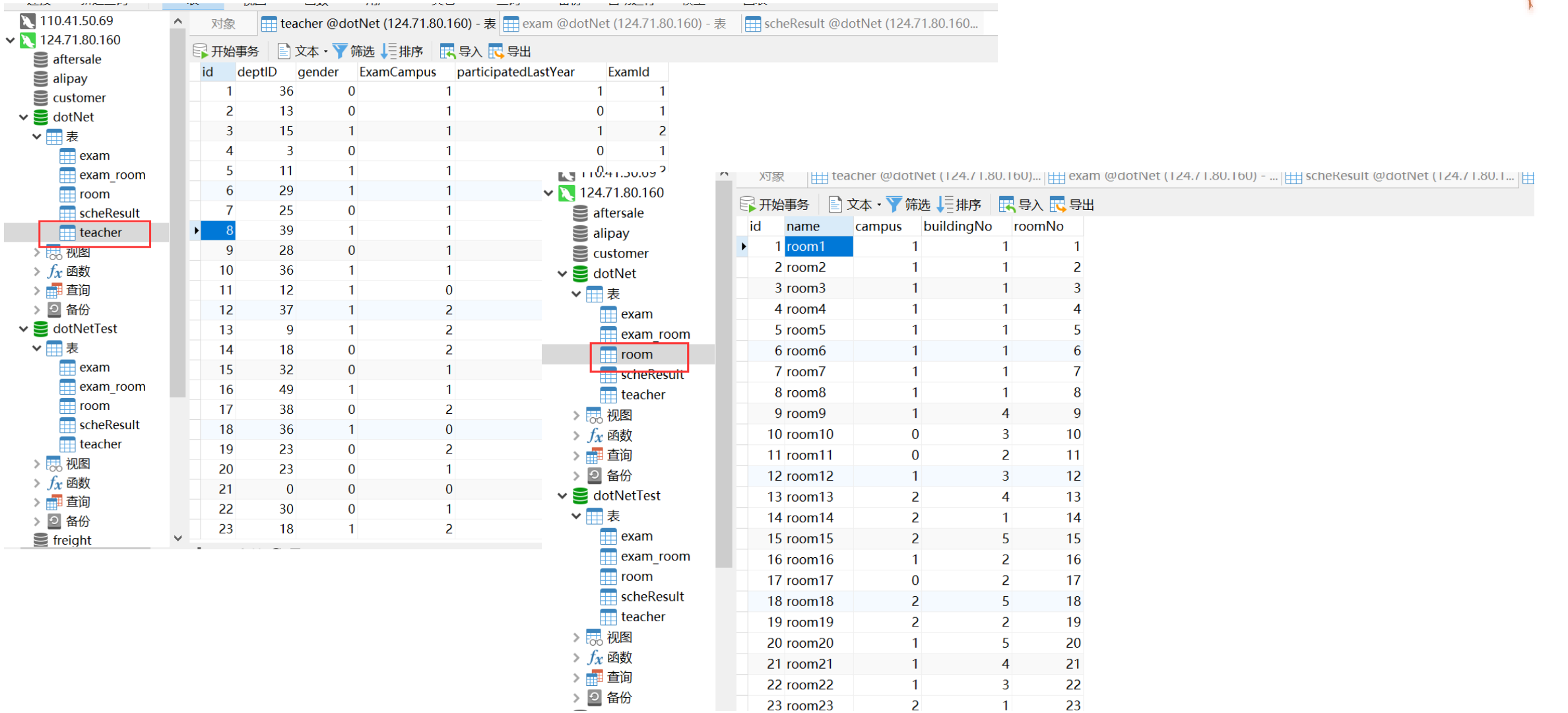
```
namespace ExamSchedulingSystem.db
{
    [Table("exam_room")]
    public class ExamRoom
    {
        [Key]
        public int id { get; set; }
        public int roomId { get; set; }
        public int examId { get; set; }
    }
}
```

02 接口设计

数据库设计
仅用到相关的表和有用的字段

```
namespace ExamSchedulingSystem
{

    public enum Result
    {
        正确分配,
        分配但有问题,
        无法分配
    }

    public class ScheResult
    {
        [Key]
        public int Id { get; set; }
        // 考试id
        public int ExamId { get; set; }

        // 考场id
        public int ExamRoomId { get; set; }

        // 监考1报名ID
        public int Invigilator1Id { get; set; }

        // 监考2报名ID
        public int Invigilator2Id { get; set; }

        // 是否正确分配
        public Result AllocationStatus { get; set; }

        // 分配规则描述
        public string AllocationRuleDescription { get; set; }
    }

}
```

02 接口设计


110.41.50.69
124.71.80.160
aftersale
alipay
customer
dotNet
表
exam
exam_room
room
scheResult
teacher
视图
函数
查询
备份
dotNetTest
表
exam
exam_room
room
scheResult
teacher
视图
函数
查询
备份
freight
对象
teacher @dotNet (124.71.80.160) - 表
exam @dotNet (124.71.80.160) - 表
scheResult @dotNet (124.71.80.160...
开始事务
文本
筛选
排序
导入
导出
id deptID gender ExamCampus participatedLastYear ExamId
1 36 0 1 1 1
2 13 0 1 0 1
3 15 1 1 1 2
4 3 0 1 0 1
5 11 1 1 0
6 29 1 1
7 25 0 1
8 39 1 1
9 28 0 1
10 36 1 1
11 12 1 0
12 37 1 2
13 9 1 2
14 18 0 2
15 32 0 1
16 49 1 1
17 38 0 2
18 36 1 0
19 23 0 2
20 23 0 1
21 0 0 0
22 30 0 1
23 18 1 2
124.71.80.160
aftersale
alipay
customer
dotNet
表
exam
exam_room
room
scheResult
teacher
视图
函数
查询
备份
dotNetTest
表
exam
exam_room
room
scheResult
teacher
视图
函数
查询
备份
开始事务
文本
筛选
排序
导入
导出
id name campus buildingNo roomNo
1 room1 1 1 1
2 room2 1 1 2
3 room3 1 1 3
4 room4 1 1 4
5 room5 1 1 5
6 room6 1 1 6
7 room7 1 1 7
8 room8 1 1 8
9 room9 1 4 9
10 room10 0 3 10
11 room11 0 2 11
12 room12 1 3 12
13 room13 2 4 13
14 room14 2 1 14
15 room15 2 5 15
16 room16 1 2 16
17 room17 0 2 17
18 room18 2 5 18
19 room19 2 2 19
20 room20 1 5 20
21 room21 1 4 21
22 room22 1 3 22
23 room23 2 1 23

03

排考算法分析

03 排考算法分析

要求

自动排考模块：

1. 每个考场的两个监考*必须*来自不同的部门；
2. 每个考场的两个监考*必须不能*是两个新老师
3. 每个考场的两个监考尽可能是老带新；
4. 每个考场的两个监考尽可能性别不同；
5. 如果监考性别比例无法男女搭配，那么尽可能的让少的那种性别分开一些
6. 在无法分配时，必须显示考场无法分配的原因。

分成三个校区开始考虑，获得每个校区的考场和老师

1. **必须条件判断：**报名老师数量 >= 需求人数 、 除最大人数部门之外的所有部门总人数不能低于需求人数的一半 、 报名的监考过的老师 >= 考场数量

2. **预处理：**随机化函数打乱考场，再遍历一次考场列表，如果前后考场连续且前考场与后考场的下一考场不连续、交换前后的考场（均匀分配）；**新老教师分组（新老原则）；部门新老分组（部门原则）**

3. 依次按照考场顺序分配老师

   3.1 **优先部门老新：**找老人最多且还有新人在其他部门的部门，尽量选择性别与上次不同的老师分配一个老师（原则5）

   3.1.1 **性别不同原则：**选择新人人数最多的不同部门的一个性别不同的新老师

   3.1.2 **次选性别相同：**选择性别相同的该部门的新老师

   3.2 **次选部门老老：**选老人最多且还有老人在其他部门的部门,尽量选择性别与上次不同的老师分配一个老师（原则5）

   3.2.1 **性别不同原则：**选择不同部门的一个性别不同的老老师

   3.2.2 **次选性别相同：**找不到性别不同、选性别相同的该部门的老老师

   3.3 **不满足部门和新老：**出现无法分配的情况，说明（两个监考必须来自不同的部门）和（两个监考必须不能是两个新老师）

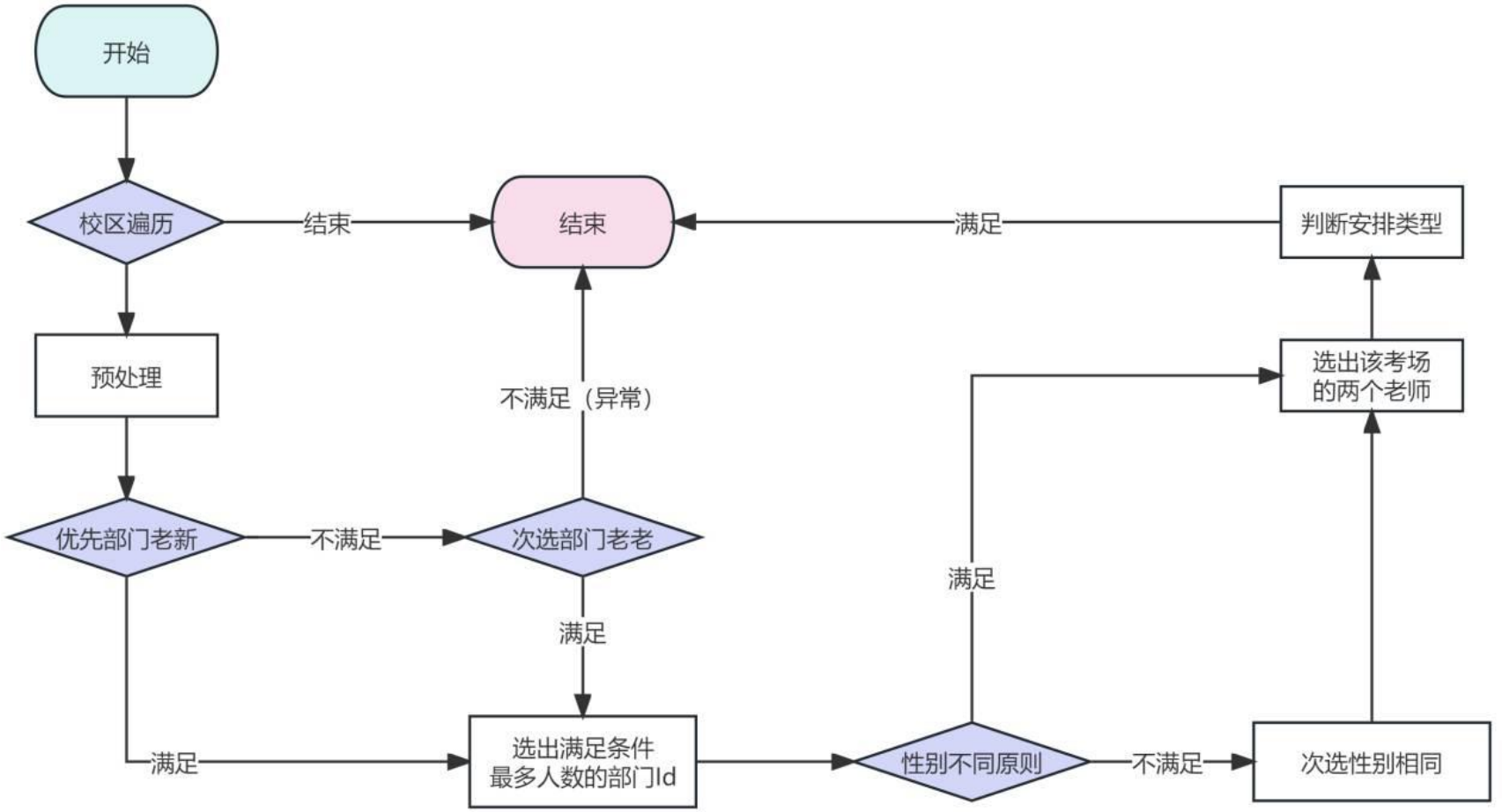
03 排考算法分析

要求

分成三个校区开始考虑，获得每个校区的考场和老师

自动排考模块：

1. 每个考场的两个监考*必须
   不同的部门；
2. 每个考场的两个监考*必须
   是两个新老师
3. 每个考场的两个监考尽可
   带新；
4. 每个考场的两个监考尽可
   不同；
5. 如果监考性别比例无法男
   那么尽可能的让少的那种
   开一些



04

排考算法设计

04 排考算法设计

分成三个校区开始考虑

```
// 每个校区分开分配
foreach(Campus campus in Enum.GetValues(typeof(Campus)))
{
    // 1 获取该校区的老师和考场
    var campusTeachers = teachers.Where(t => t.ExamCampus == campus).ToList();
    var campusRooms = rooms.Where(r => r.campus == campus).ToList();

    string res = canSchedule(campusTeachers,campusRooms);

    if(res!=""){
        // scheResults.Clear();
        scheResults.Add(new ScheResult{
            AllocationStatus = Result.无法分配,
            AllocationRuleDescription = campus.ToString() + "无法分配, " + res
        });
        return scheResults;
    }
```

```
public string canSchedule(List<Teacher> teachers, List<Room> rooms)
{
    // 1 判断必须的条件
    if (rooms == null || teachers == null)
    {
        return "考场或老师为空";
    }
    // 2 报名老师数量>=需求人数
    if (teachers.Count < rooms.Count * 2)
    {
        return "报名老师数量不足";
    }
    // 3 除最大人数部门之外的所有部门总人数不能低于需求人数的一半
    int maxDeptTeacherCount = 0;
    foreach (int deptId in teachers.Select(t => t.DeptId).Distinct().ToList())
    {
        int deptTeacherCount = teachers.Where(t => t.DeptId == deptId).Count();
        if (deptTeacherCount > maxDeptTeacherCount)
        {
            maxDeptTeacherCount = deptTeacherCount;
        }
    }
    int otherDeptTeacherCount = teachers.Count - maxDeptTeacherCount;
    if (otherDeptTeacherCount < rooms.Count)
    {
        return "除最大人数部门之外的所有部门总人数不足";
    }
    // 4 报名的监考过的老师>=考场数量
    var participatedTeachers = teachers.Where(t => t.ParticipatedLastYear == true).ToList();
    if (participatedTeachers.Count < rooms.Count)
    {
        return "报名的监考过的老师数量不足";
    }
    return "";
}
```

1. **必须条件判断：**报名老师数量 >= 需求人数 + 除最大人数部门之外的所有部门总人数不能低于需求人数的一半 + 报名的监考过的老师 >= 考场数量

04 排考算法设计

2. **预处理：**随机化函数打乱考场，再遍历一次考场列表，如果前后考场连续且前考场与后考场的下一考场不连续、交换前后的考场（均匀分配）；**新老教师分组（新老原则）；部门新老分组（部门原则）**

```
// 2 随机打乱考场
Random random = new Random();
campusRooms = campusRooms.OrderBy(r => random.Next()).ToList();

// 再遍历一次考场列表、如果前后考场考室连续且前考场与后考场的下一考场不连续、交换前后的考场
for(int i = 0; i < campusRooms.Count - 1; i++)
{
    if(campusRooms[i].buildingNo == campusRooms[i + 1].buildingNo && campusRooms[i].roomNo + 1 == campusRooms[i + 1].roomNo)
    {
        if(i + 2 < campusRooms.Count && campusRooms[i + 1].buildingNo != campusRooms[i + 2].buildingNo)
        {
            var temp = campusRooms[i];
            campusRooms[i] = campusRooms[i + 1];
            campusRooms[i + 1] = temp;
        }
    }
}
```

04 排考算法设计

2. **预处理：**随机化函数打乱考场，再遍历一次考场列表，如果前后考场连续且前考场与后考场的下一考场不连续、交换前后的考场（均匀分配）；**新老教师分组（新老原则）；部门新老分组（部门原则）**

```
// 3 数据准备
int alreadyAllocatedTeacherCount = 0;
int lastGender = 0;

// 把老师分成新老两组老师
List<Teacher> newTeachers = campusTeachers.Where(t => t.ParticipatedLastYear == false).ToList();
List<Teacher> oldTeachers = campusTeachers.Where(t => t.ParticipatedLastYear == true).ToList();

Dictionary<int, int> newDeptTeacherCount = new Dictionary<int, int>();
Dictionary<int, int> oldDeptTeacherCount = new Dictionary<int, int>();
// 分别记录两组老师的不同部门的剩余老师数量
foreach (int deptId in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
{
    newDeptTeacherCount.Add(deptId, newTeachers.Where(t => t.DeptId == deptId).Count());
    oldDeptTeacherCount.Add(deptId, oldTeachers.Where(t => t.DeptId == deptId).Count());
}
```

04 排考算法设计

3. 依次按照考场顺序分配老师

3.1 **优先部门老新：**找老人最多且还有新人在其他部门的部门，尽量选择性别与上次不同的老师分配一个老师（原则5）

```
// 4 依次为考场分配老师
while (alreadyAllocatedTeacherCount != campusRooms.Count())
{

    // 4.1 找老人最多且还有新人在其他部门的部门
    int maxOldDeptId = 0;
    int maxOldDeptTeacherCount = 0;
    int solutionType = 0;
    foreach (int deptId in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
    {
        if(oldDeptTeacherCount[deptId] > maxOldDeptTeacherCount)
        {
            foreach (int deptId2 in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
            {
                if(newDeptTeacherCount[deptId2] > 0 && deptId2 != deptId)
                {
                    maxOldDeptId = deptId;
                    maxOldDeptTeacherCount = oldDeptTeacherCount[deptId];
                    break;
                }
            }
        }
    }
```

04 排考算法设计

3.2 **次选部门老老：**选老人最多且还有老人在其他部门的部门,尽量选择性别与上次不同的老师分配一个老师（原则5）

3.3 **不满足部门和新老：**出现无法分配的情况，说明（两个监考必须来自不同的部门）和（两个监考必须不能是两个新老师）

```
// 4.2 次优解，选老人最多且还有老人在其他部门的部门
if (maxOldDeptId == 0)
{
    solutionType = 1;
    foreach (int deptId in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
    {
        if(oldDeptTeacherCount[deptId] > maxOldDeptTeacherCount)
        {
            foreach (int deptId2 in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
            {
                if(oldDeptTeacherCount[deptId2] > 0 && deptId2 != deptId)
                {
                    maxOldDeptId = deptId;
                    maxOldDeptTeacherCount = oldDeptTeacherCount[deptId];
                    break;
                }
            }
        }
    }
}

// 4.3 无解
if (maxOldDeptId == 0)
{
    // scheResults.Clear();
    scheResults.Add(new ScheResult{
        AllocationStatus = Result.无法分配,
        AllocationRuleDescription = campus.ToString() + "无法分配，  " + "无法再找到部门不同的老师，还差" + (campusRooms.Count - alreadyAll
    });
    break;
}
```

04 排考算法设计

3.1 优先部门老新：找老人最多且还有新人在其他部门的部门，尽量选择性别与上次不同的老师分配一个老师（原则5）

3.1.1 **性别不同原则：**选择新人人数最多的不同部门的一个性别不同的新老师

3.1.2 **次选性别相同：**选择性别相同的该部门的新老师

```
// 选出考场
ScheResult scheResult = new ScheResult();
scheResult.ExamId = ExamId;
scheResult.ExamRoomId = campusRooms[alreadyAllocatedTeacherCount].getid();

// 找到该部门的所有监考过的老师
List<Teacher> maxOldDeptTeachers = oldTeachers.Where(t => t.DeptId == maxOldDeptId).ToList();

// 4.1.1 & 4.2.1从该部门的老师中尽量找到一个性别和lastGender不同的老师
Teacher select1 = maxOldDeptTeachers[0];
Teacher select2 = new Teacher();
foreach(Teacher t in maxOldDeptTeachers){
    if(t.Gender != lastGender){
        select1 = t;
        break;
    }
}
scheResult.Invigilator1Id = select1.getid();
lastGender = select1.Gender;
// 移出 oldDeptTeacherCount 和 oldTeachers
oldDeptTeacherCount[maxOldDeptId]--;
oldTeachers.Remove(select1);
```

04 排考算法设计

3.1 优先部门老新：找老人最多且还有新人在其他部门的部门，尽量选择性别与上次不同的老师分配一个老师（原则5）

3.1.1 **性别不同原则：**选择新人人数最多的不同部门的一个性别不同的新老师

3.1.2 **次选性别相同：**选择性别相同的该部门的新老师

```
if(solutionType == 0){
    // 4.1.2 找新人人数最多的不同部门
    int maxNewDeptId = 0;
    int maxNewDeptTeacherCount = 0;
    foreach (int deptId in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
    {
        if(newDeptTeacherCount[deptId] > maxNewDeptTeacherCount && deptId != maxOldDeptId)
        {
            maxNewDeptId = deptId;
            maxNewDeptTeacherCount = newDeptTeacherCount[deptId];
        }
    }

    // 找到该部门的所有没监考过的老师
    List<Teacher> maxNewDeptTeachers = newTeachers.Where(t => t.DeptId == maxNewDeptId).ToList();

    // 从该部门的老师中尽量找到一个性别和select1不同的老师
    select2 = maxNewDeptTeachers[0];
    foreach(Teacher t in maxNewDeptTeachers){
        if(t.Gender != select1.Gender){
            select2 = t;
            break;
        }
    }
    scheResult.Invigilator2Id = select2.getid();
    // 移出 newDeptTeacherCount 和 newTeachers
    newDeptTeacherCount[maxNewDeptId]--;
    newTeachers.Remove(select2);
```

04 排考算法设计

3.2 **次选部门老老：**选老人最多且还有老人在其他部门的部门,尽量选择性别与上次不同的老师分配一个老师（原则5）

3.2.1 **性别不同原则：**选择不同部门的一个性别不同的老老师

3.2.2 **次选性别相同：**找不到性别不同、选性别相同的该部门的老老师

```
}else{
    // 4.2.2找老人人数最多的不同部门
    int maxOldDeptId2 = 0;
    int maxOldDeptTeacherCount2 = 0;
    foreach (int deptId in campusTeachers.Select(t => t.DeptId).Distinct().ToList())
    {
        if(oldDeptTeacherCount[deptId] > maxOldDeptTeacherCount2 && deptId != maxOldDeptId)
        {
            maxOldDeptId2 = deptId;
            maxOldDeptTeacherCount2 = oldDeptTeacherCount[deptId];
        }
    }

    // 找到该部门的所有的老师
    List<Teacher> maxOldDeptTeachers2 = oldTeachers.Where(t => t.DeptId == maxOldDeptId2).ToList();

    // 从该部门的老师中尽量找到一个性别和select1不同的老师
    select2 = maxOldDeptTeachers2[0];
    foreach(Teacher t in maxOldDeptTeachers2){
        if(t.Gender != select1.Gender){
            select2 = t;
            break;
        }
    }
    scheResult.Invigilator2Id = select2.getid();
    // 移出 oldDeptTeacherCount 和 oldTeachers
    oldDeptTeacherCount[maxOldDeptId2]--;
    oldTeachers.Remove(select2);

}
```

04 排考算法设计

最后得出每个考场的安排结果

```
    // 4.1.3 & 4.2.3添加到结果
    if(select1.Gender == select2.Gender){
        scheResult.AllocationStatus = Result.分配但有问题;
        scheResult.AllocationRuleDescription = campus.ToString() + "错误分配, " + "老师 " + scheResult.Invigilator1Id + " 和 " + scheResult.Invigilator2Id + " 性别相同";
        scheResults.Add(scheResult);
    }else if(select1.ParticipatedLastYear == true && select2.ParticipatedLastYear == true){
        scheResult.AllocationStatus = Result.分配但有问题;
        scheResult.AllocationRuleDescription = campus.ToString() + "错误分配, " + "老师 " + scheResult.Invigilator1Id + " 和 " + scheResult.Invigilator2Id + " 都监考过";
        scheResults.Add(scheResult);
    }else{
        scheResult.AllocationStatus = Result.正确分配;
        scheResult.AllocationRuleDescription = campus.ToString() + "已分配考场 " + scheResult.ExamRoomId + " 给老师 " + scheResult.Invigilator1Id + " 和 " + scheResult.Invigilator2Id +
                                               " 监考, 该考场为 " + scheResult.ExamRoomId + " 号考场, 老师的部门分别为 " + select1.DeptId + " 和 " + select2.DeptId +
                                               " 老师的性别分别为 " + select1.Gender + " 和 " + select2.Gender + " .";

        scheResults.Add(scheResult);
    }
    alreadyAllocatedTeacherCount++;
}
```

考场分配表:

- 分配记录ID
- 考试ID
- 考场ID
- 监考1报名ID
- 监考2报名ID
- 是否正确分配（枚举：正确分配，分配但有问题，无法分配）
- 分配规则描述（字符串存一个JSON，表述无法分配时，是由于上述5条规则无法处理）

04 排考算法设计

函数的时间复杂度主要取决于几个部分：

Distinct()和ToList()操作：这两个操作的时间复杂度都是O(n)，其中n是列表的长度。

foreach循环：这个循环的时间复杂度是O(n)，其中n是Campus枚举的成员数量。在这个循环中，有一些操作的时间复杂度是O(m)，其中m是teachers或rooms的长度。

Where()和ToList()操作：这两个操作的时间复杂度都是O(m)，其中m是teachers或rooms的长度。

OrderBy()操作：这个操作的时间复杂度是O(m log m)，其中m是rooms的长度。

for循环：这个循环的时间复杂度是O(m)，其中m是rooms的长度。

while循环：这个循环的时间复杂度是O(m)，其中m是rooms的长度。在这个循环中，有一些操作的时间复杂度是O(n)，其中n是teachers的长度。

综上所述，这个函数的总体时间复杂度是 $O(n*m + m \log m)$，其中n是teachers的长度，m是rooms的长度。

05

数据测试

05 测试一：必须要老带老

POST https://localhost:7205/scheduling/7 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (4) Test Results Status: 200 OK Time: 254 ms Size: 1.03 KB Save as example

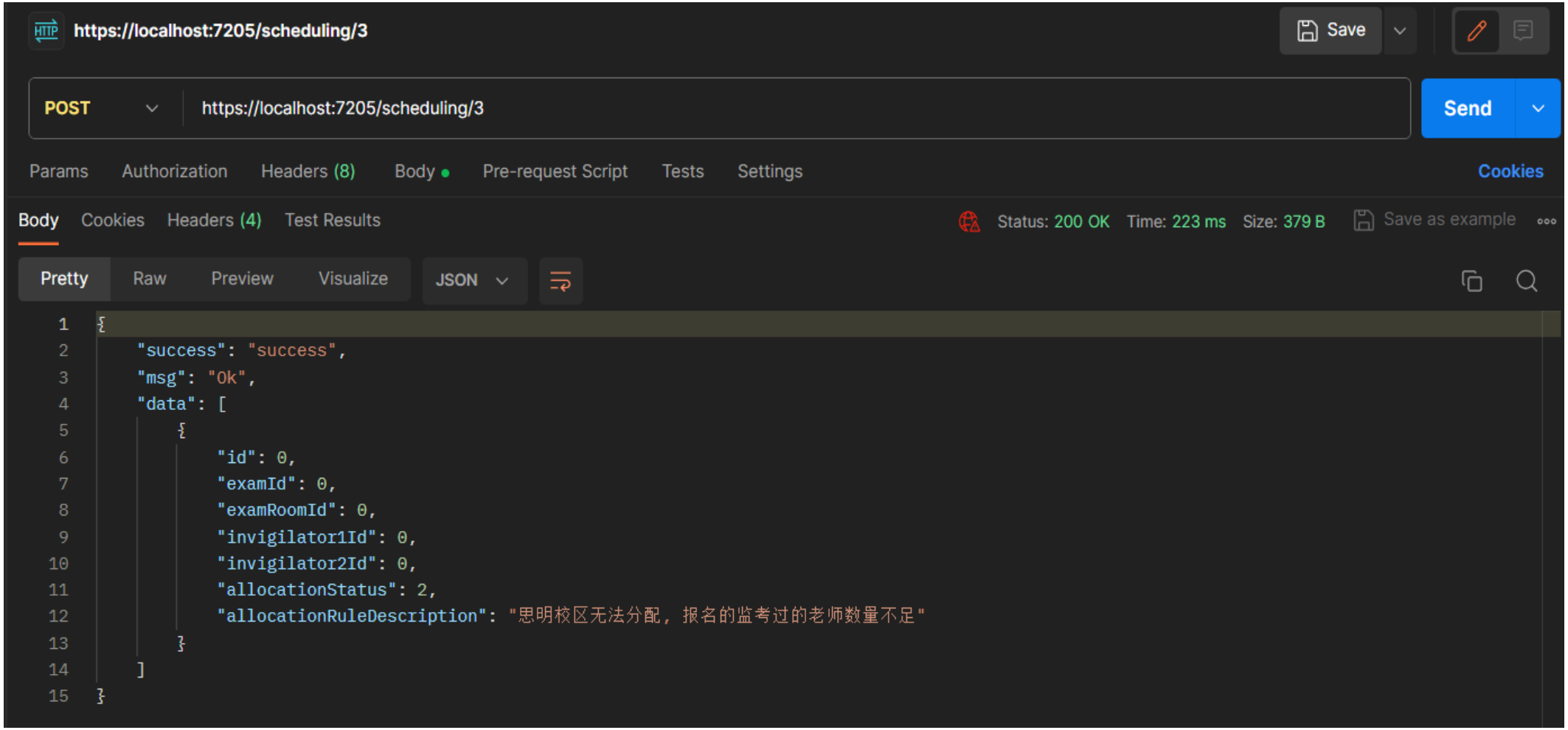
Pretty Raw Preview Visualize JSON


```
    "data": [
        {
            "id": 1,
            "examId": 7,
            "examRoomId": 274,
            "invigilator1Id": 3000,
            "invigilator2Id": 3005,
            "allocationStatus": 0,
            "allocationRuleDescription": "思明校区已分配考场 274 给老师 3000 和 3005 监考，该考场为 274 号考场，老师的部门分别为 36 和 1 老师的性别分别为 1 和 0 ."
        },
        {
            "id": 2,
            "examId": 7,
            "examRoomId": 273,
            "invigilator1Id": 3001,
            "invigilator2Id": 3002,
            "allocationStatus": 1,
            "allocationRuleDescription": "思明校区错误分配，老师 3001 和 3002 都监考过"
        },
        {
            "id": 3,
            "examId": 7,
            "examRoomId": 272,
            "invigilator1Id": 3003,
            "invigilator2Id": 3004,
            "allocationStatus": 1,
            "allocationRuleDescription": "思明校区错误分配，老师 3003 和 3004 都监考过"
```

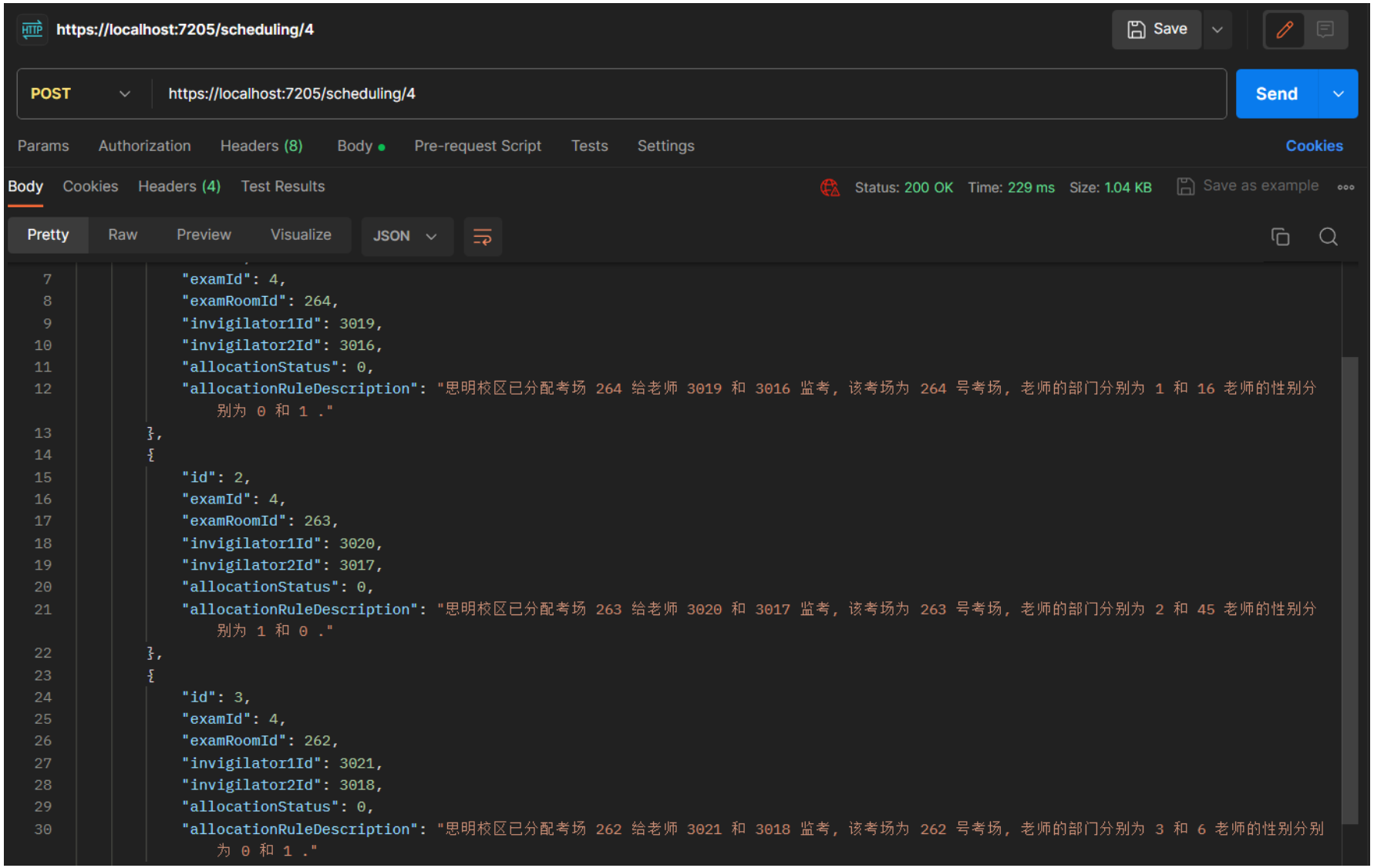

后面三个考场提示出现问题

05 测试二：有新带新（无老）

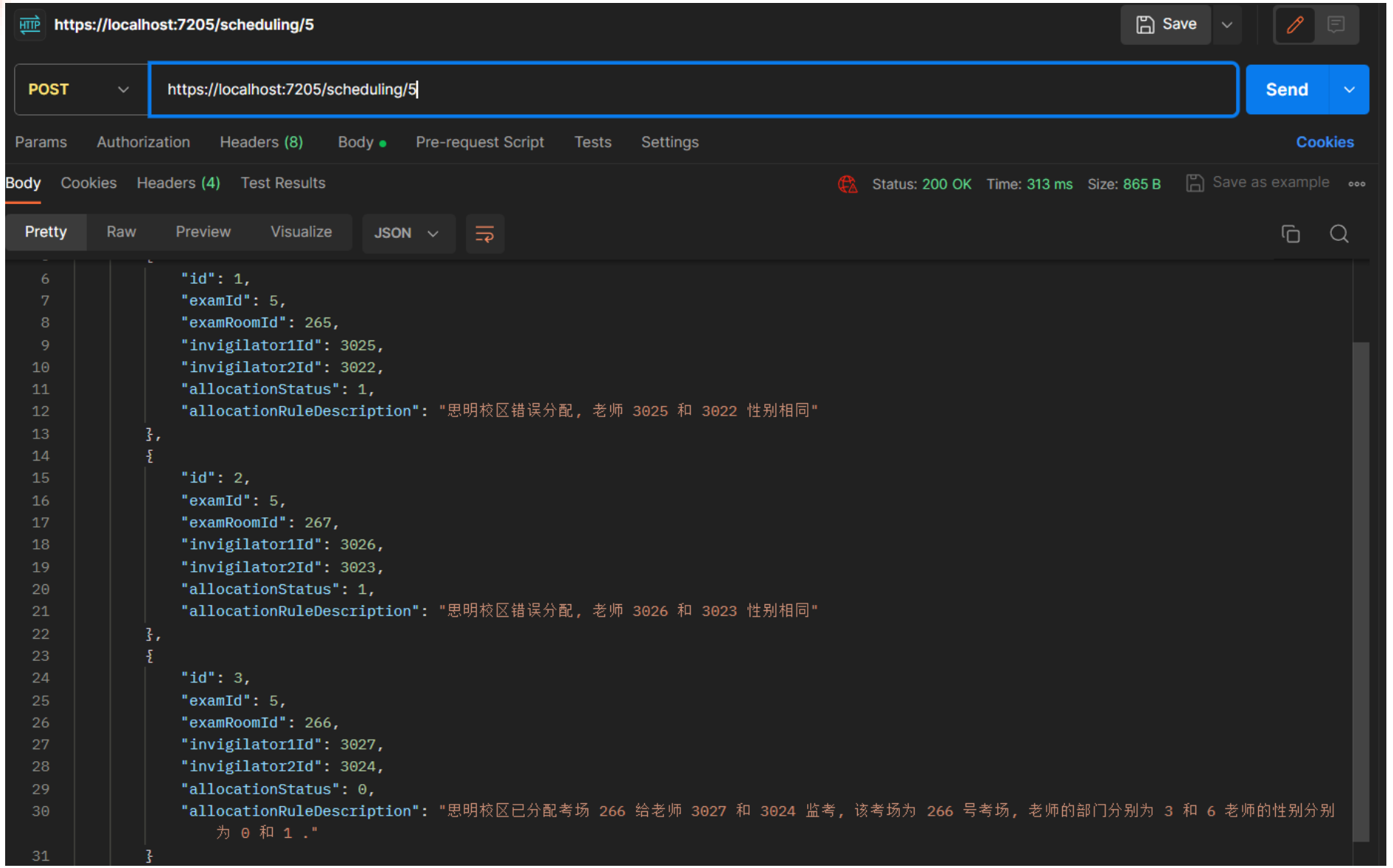
对应考场提示无法排考



05 测试三：没有老带老和新带新..(正常)



05 测试四：必须有性别相同



对应的考场会输出错误信息，但是会给出排考的数据

05 测试五：没有性别相同

https://localhost:7205/scheduling/4

Save

POST https://localhost:7205/scheduling/4 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Body Cookies Headers (4) Test Results Status: 200 OK Time: 229 ms Size: 1.04 KB Save as example

Pretty Raw Preview Visualize JSON


```
        "examId": 4,
        "examRoomId": 264,
        "invigilator1Id": 3019,
        "invigilator2Id": 3016,
        "allocationStatus": 0,
        "allocationRuleDescription": "思明校区已分配考场 264 给老师 3019 和 3016 监考，该考场为 264 号考场，老师的部门分别为 1 和 16 老师的性别分别为 0 和 1 ."
    },
    {
        "id": 2,
        "examId": 4,
        "examRoomId": 263,
        "invigilator1Id": 3020,
        "invigilator2Id": 3017,
        "allocationStatus": 0,
        "allocationRuleDescription": "思明校区已分配考场 263 给老师 3020 和 3017 监考，该考场为 263 号考场，老师的部门分别为 2 和 45 老师的性别分别为 1 和 0 ."
    },
    {
        "id": 3,
        "examId": 4,
        "examRoomId": 262,
        "invigilator1Id": 3021,
        "invigilator2Id": 3018,
        "allocationStatus": 0,
        "allocationRuleDescription": "思明校区已分配考场 262 给老师 3021 和 3018 监考，该考场为 262 号考场，老师的部门分别为 3 和 6 老师的性别分别为 0 和 1 ."
```

05 测试六：大数据集

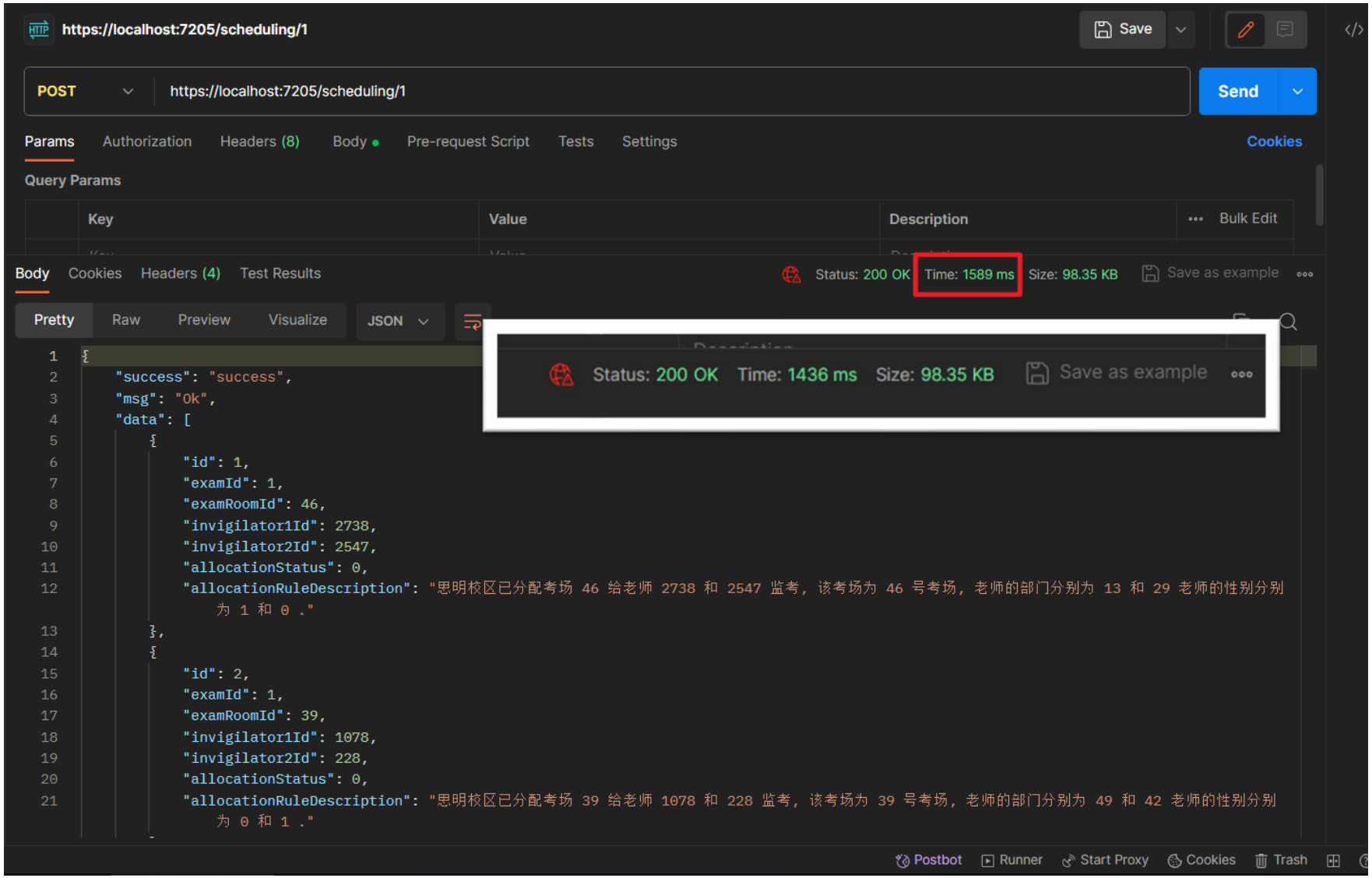
14997	33	1	0	0	1
14998	165	0	1	0	1
14999	4	1	1	0	1
15000	15	0	1	0	1
15001	1	1	1	1	3
15002	2	1	1	1	3
15003	3	1	1	1	3
15004	4	1	1	1	3
15005	5	1	1	1	3
15006	6	1	1	0	3
15007	7	1	1	0	3
15008	8	1	1	0	3

490	room490	0	2	490
491	room491	0	5	491
492	room492	0	4	492
493	room493	1	5	493
494	room494	2	4	494
495	room495	1	3	495
496	room496	1	1	496
497	room497	0	1	497
498	room498	1	2	498
499	room499	0	1	499
500	room500	2	3	500

MySQL数据库中报名的老师数量为1w5人，考场个数为500个，程序运行的时候实际考试id=1对应的考场数和老师数通过日志输出分别是496个和13446个（因为存在部分老师和考场对应的考试id不是1）

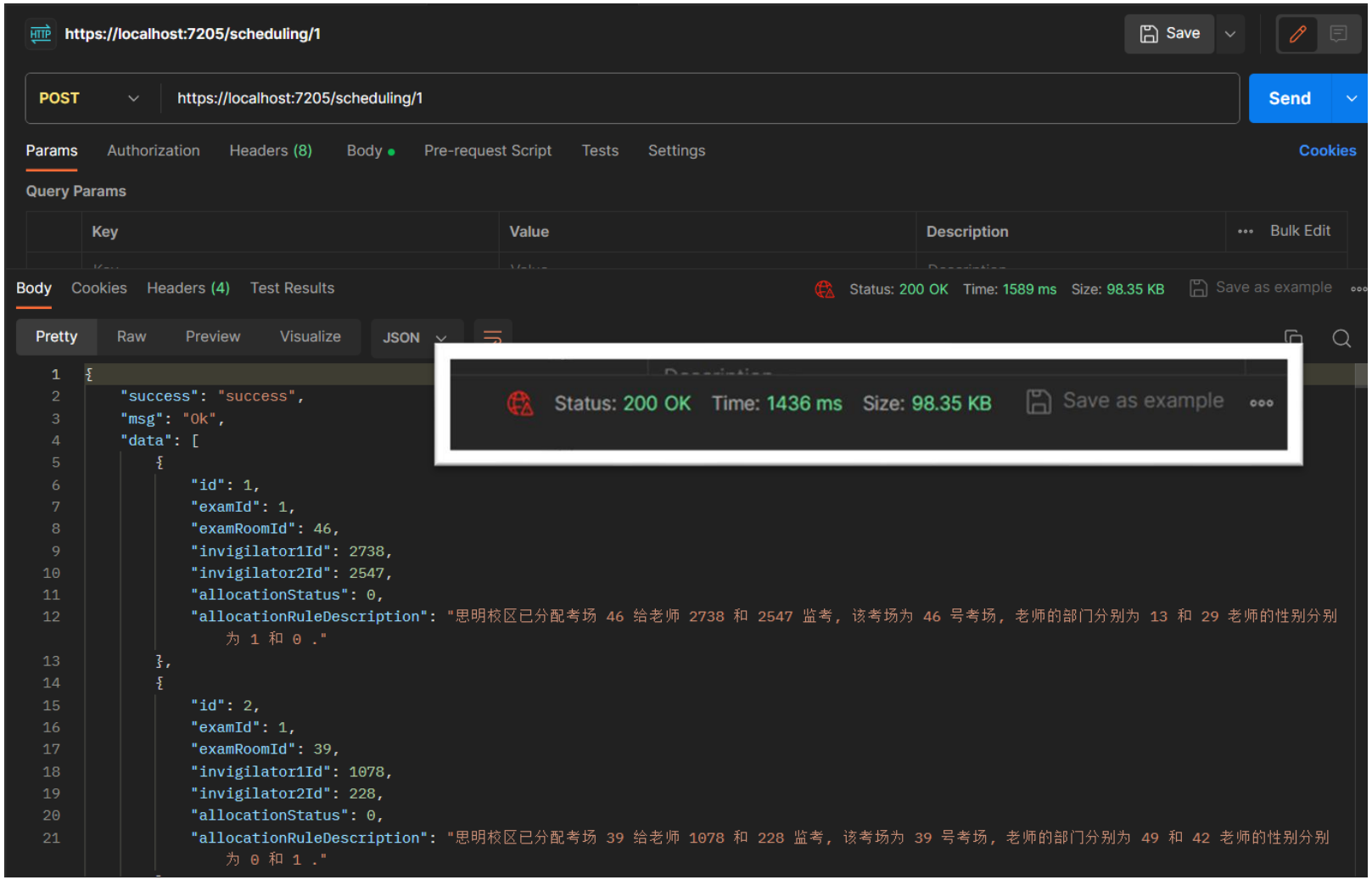
```
      Content root path: D:\Desktop\ExamSchedulingSystem\
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (60ms) [Parameters=[@__id_0='?' (DbType = Int32)], CommandType='Text', CommandTimeout='30']
      SELECT `r`.`id`, `r`.`buildingNo`, `r`.`campus`, `r`.`name`, `r`.`roomNo`
      FROM `room` AS `r`
      INNER JOIN `exam_room` AS `e` ON `r`.`id` = `e`.`roomId`
      WHERE `e`.`examId` = @__id_0
rooms:496
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (29ms) [Parameters=[@__id_0='?' (DbType = Int32)], CommandType='Text', CommandTimeout='30']
      SELECT `t`.`id`, `t`.`DeptId`, `t`.`ExamCampus`, `t`.`ExamId`, `t`.`Gender`, `t`.`ParticipatedLastYear`
      FROM `teacher` AS `t`
      WHERE `t`.`ExamId` = @__id_0
teachers:13446
```

05 测试六：大数据集



对于496个考场和13446个报名老师，可以发现程序在1.5s左右完成，扣除掉网络开销和数据库开销，证明算法的效率很高

05 测试六：大数据集



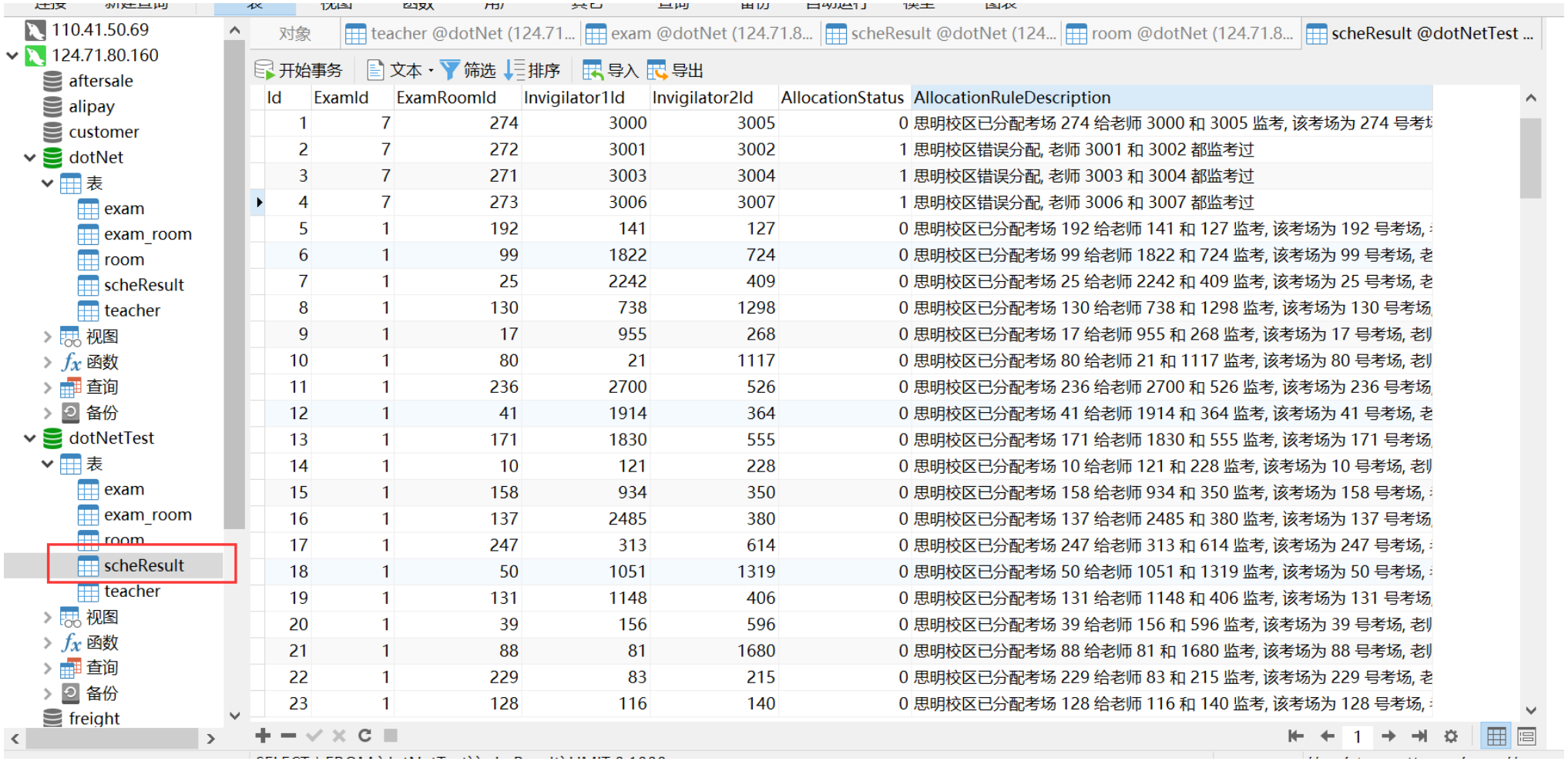
对于496个考场和13446个报名老师，数据库有缓存机制，第二次请求的时候速度会更快

以上对应的test：

id=7老带老　id=3新带新　id=4没有老戴老、新带新、性别都正常 id=5有性别相同 id=1大数据集

05 测试六：大数据集

结果算出后自动存入数据库



Id	ExamId	ExamRoomId	Invigilator1Id	Invigilator2Id	AllocationStatus	AllocationRuleDescription
1	7	274	3000	3005	0	思明校区已分配考场 274 给老师 3000 和 3005 监考, 该考场为 274 号考场
2	7	272	3001	3002	1	思明校区错误分配, 老师 3001 和 3002 都监考过
3	7	271	3003	3004	1	思明校区错误分配, 老师 3003 和 3004 都监考过
4	7	273	3006	3007	1	思明校区错误分配, 老师 3006 和 3007 都监考过
5	1	192	141	127	0	思明校区已分配考场 192 给老师 141 和 127 监考, 该考场为 192 号考场,
6	1	99	1822	724	0	思明校区已分配考场 99 给老师 1822 和 724 监考, 该考场为 99 号考场, 老
7	1	25	2242	409	0	思明校区已分配考场 25 给老师 2242 和 409 监考, 该考场为 25 号考场, 老
8	1	130	738	1298	0	思明校区已分配考场 130 给老师 738 和 1298 监考, 该考场为 130 号考场
9	1	17	955	268	0	思明校区已分配考场 17 给老师 955 和 268 监考, 该考场为 17 号考场, 老
10	1	80	21	1117	0	思明校区已分配考场 80 给老师 21 和 1117 监考, 该考场为 80 号考场, 老
11	1	236	2700	526	0	思明校区已分配考场 236 给老师 2700 和 526 监考, 该考场为 236 号考场
12	1	41	1914	364	0	思明校区已分配考场 41 给老师 1914 和 364 监考, 该考场为 41 号考场, 老
13	1	171	1830	555	0	思明校区已分配考场 171 给老师 1830 和 555 监考, 该考场为 171 号考场
14	1	10	121	228	0	思明校区已分配考场 10 给老师 121 和 228 监考, 该考场为 10 号考场, 老
15	1	158	934	350	0	思明校区已分配考场 158 给老师 934 和 350 监考, 该考场为 158 号考场,
16	1	137	2485	380	0	思明校区已分配考场 137 给老师 2485 和 380 监考, 该考场为 137 号考场
17	1	247	313	614	0	思明校区已分配考场 247 给老师 313 和 614 监考, 该考场为 247 号考场,
18	1	50	1051	1319	0	思明校区已分配考场 50 给老师 1051 和 1319 监考, 该考场为 50 号考场,
19	1	131	1148	406	0	思明校区已分配考场 131 给老师 1148 和 406 监考, 该考场为 131 号考场
20	1	39	156	596	0	思明校区已分配考场 39 给老师 156 和 596 监考, 该考场为 39 号考场, 老
21	1	88	81	1680	0	思明校区已分配考场 88 给老师 81 和 1680 监考, 该考场为 88 号考场, 老
22	1	229	83	215	0	思明校区已分配考场 229 给老师 83 和 215 监考, 该考场为 229 号考场, 老
23	1	128	116	140	0	思明校区已分配考场 128 给老师 116 和 140 监考, 该考场为 128 号考场,