软件体系结构与设计
考查报告

姓　　名：焦熙鹏
学　　号：20231002234
班　　级：191231
学　　院：计算机学院
题　　目：使用 JMeter 测试 ASP.NET Core CRUD 页面性能
指导 教师：王勇

2025 年 12 月

目录

第 1 章 引言

1.1 实验背景

随着 Web 应用的广泛使用，系统性能已成为衡量软件质量的重要指标之一。在实际开发中，不仅要保证功能的正确性，还需要确保系统在高并发场景下的稳定性和响应速度。本实验基于 ASP.NET Core 框架开发了一个学生管理系统，并使用 Apache JMeter 进行性能测试，以评估系统在不同负载下的表现。

1.2 实验目的

1. 掌握 ASP.NET Core Web 应用的开发流程
2. 实现完整的 CRUD（创建、读取、更新、删除）功能
3. 学习使用 Entity Framework Core 进行数据持久化
4. 掌握 Apache JMeter 性能测试工具的使用方法
5. 分析 Web 应用的性能指标，识别性能瓶颈
6. 理解并发访问对系统性能的影响

1.3 实验内容

本实验主要包含两个部分：

实验内容之一：Web 应用开发

使用 ASP.NET Core Razor Pages 开发学生管理系统，实现对学生信息的完整 CRUD 操作，使用 Entity Framework Core 进行数据库操作，实现数据验证和错误处理

实验内容之二：JMeter 性能测试

设计针对 CRUD 操作的性能测试场景，配置多线程并发测试，然后收集和分析性能指标数据

第 2 章 实验环境与工具

2.1 开发环境

操作系统： Windows

开发框架： .NET 8.0

开发工具： Visual Studio Code / Visual Studio 2022

数据库： SQL Server Express (localhost\SQLEXPRESS)

Web 服务器： Kestrel (http://localhost:5110)

2.2 测试工具

性能测试工具： Apache JMeter

测试脚本： StudentApp_Performance_Test.jmx

测试数据： student_ids.csv

并发线程数： 100 线程/操作

总样本数： 500 个请求

2.3 技术栈

后端框架： ASP.NET Core 8.0

页面技术： Razor Pages

ORM 框架： Entity Framework Core

数据库提供程序： Microsoft.EntityFrameworkCore.SqlServer

前端框架： Bootstrap 5

数据验证： Data Annotations

数据库迁移： EF Core Migrations

第 3 章 系统设计与实现

3.1 系统架构

学生管理系统采用经典的三层架构:

```
┌─────────────────────────────────────┐
│     表示层 (Razor Pages)            │
│  - Index.cshtml                     │
│  - Create.cshtml                    │
│  - Edit.cshtml                      │
│  - Details.cshtml                   │
│  - Delete.cshtml                    │
└─────────────────────────────────────┘
            ↓
┌─────────────────────────────────────┐
│     业务逻辑层 (PageModel)          │
│  - Index.cshtml.cs                  │
│  - Create.cshtml.cs                 │
│  - Edit.cshtml.cs                   │
│  - Details.cshtml.cs                │
│  - Delete.cshtml.cs                 │
└─────────────────────────────────────┘
            ↓
┌─────────────────────────────────────┐
```

```
┌──────────────────────────────────────┐
│   数据访问层 (EF Core)               │
│  - ApplicationDbContext              │
│  - Student Model                     │
└──────────────────────────────────────┘
                ↓
┌──────────────────────────────────────┐
│   数据库 (SQL Server)                │
└──────────────────────────────────────┘
```

3.2 数据库设计

Student 表结构：

字段名	数据类型	说明	约束
Id	int	学生 ID	主键，自增
Name	string	姓名	必填
Age	int	年龄	必填
Email	string	电子邮件	必填，唯一
RegistrationDate	DateTime	注册日期	默认当前日期

3.3 CRUD 功能实现

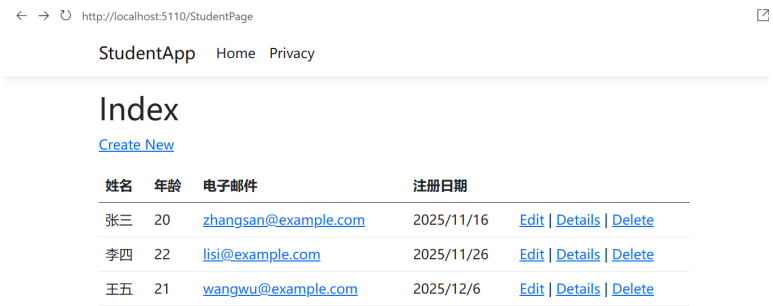

图 3-1

如图 3-1，系统在本地服务器（http://localhost:5110）上的正常运行。该页面是系统的核心入口，直观体现了数据读取（Read）功能的成功实现。

界面功能解析：

数据展示： 页面通过表格形式动态加载并显示了数据库 Students 表中的所有记录，包含姓名、年龄、电子邮件及注册日期等关键字段。

操作导航：每一行数据后侧均提供了标准的操作入口——编辑 (Edit)、详情 (Details) 和删除 (Delete)，顶部的 Create New 链接则用于跳转至新增页面。这证明了系统前端（Razor Pages）与后端（EF Core）的数据交互已完全打通。

第 4 章 性能测试设计

4.1 被测系统重点

本次性能测试的关键测试对象和重点如下：

页面： StudentPage (Index) & Create

索引页面：/StudentPage - 展示学生列表

创建页面：/StudentPage/Create - 添加新学生表单

功能： 展示 Students 列表 & 添加新学生

Index 页面功能： 从数据库查询所有学生记录，在页面上以表格形式展示，包含姓名、年龄、电子邮件、注册日期等信息

Create 页面功能： 提供表单接收用户输入，验证数据有效性，将新学生信息保存到数据库

技术： Razor Pages + EF Core + SQL Server

前端技术： ASP.NET Core Razor Pages 模板引擎

ORM 框架： Entity Framework Core 8.0

数据库： SQL Server Express (localhost\SQLEXPRESS)

✓ 测试重点： 高并发访问 Index 与 Create 页的响应情况

并发场景： 100 个并发用户同时访问

测试目标：

关注指标： 平均响应时间、最大/最小响应时间、错误率、吞吐量

4.2 测试场景设计

本次测试模拟 100 个并发用户同时访问系统，执行 CRUD 操作的场景。测试组包含 5 个主要操作:

测试操作	描述	请求类型	样本数	测试重点
查询学生列表	获取学生索引页面（Index）	GET	100	**✓ 高并发读取性能**
创建新学生	提交新学生表单（Create）	POST	100	**✓ 高并发写入性能**
查看学生详情	查看特定学生信息	GET	100	
编辑学生信息	更新学生数据	POST	100	
删除学生	删除学生记录	POST	100	

4.2 测试参数配置

线程组配置：

线程数（并发用户数）：100

循环次数：1

Ramp-Up 时间：根据需要调整

总请求数：500

HTTP 请求配置：

服务器：localhost

端口：5110

协议：HTTP

4.3 测试数据准备

使用 SeedTestData.sql 脚本预先填充测试数据，student_ids.csv 文件提供用于查询、编辑和删除操作的学生 ID 列表，确保测试过程中有足够的数据支持。

第 5 章 测试结果与分析

5.1 测试执行概况

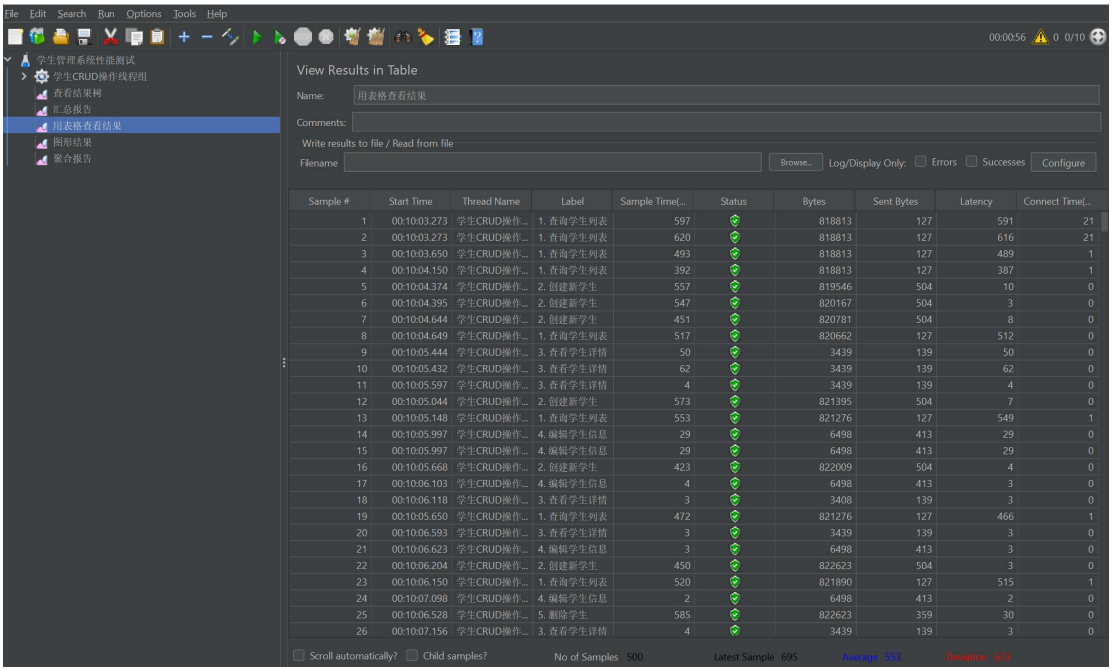

图 5-1 测试运行情况

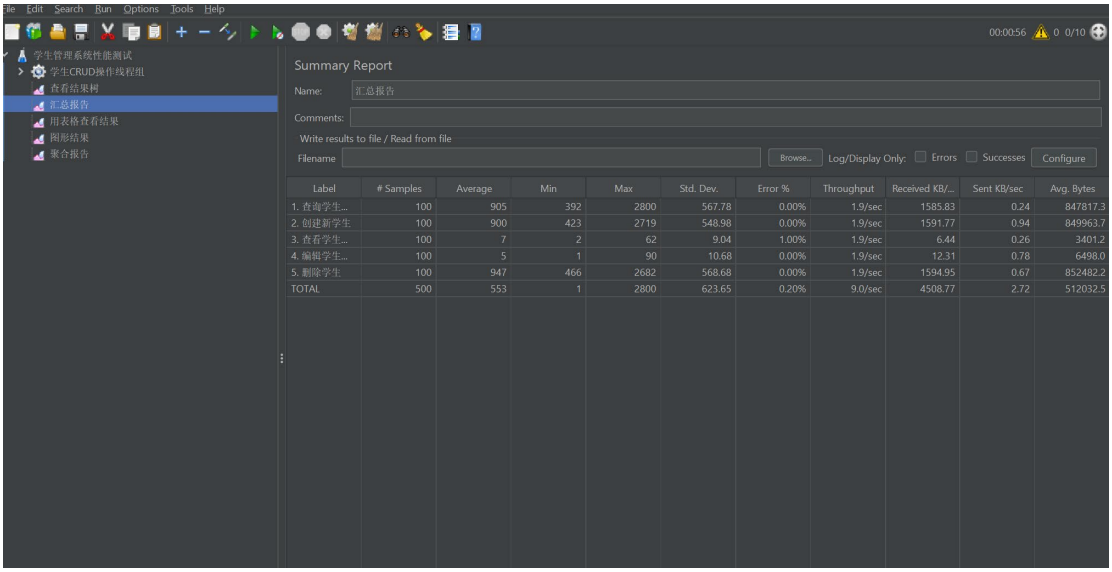

图 5-2 测试汇总报告

如图 5-1，测试被成功运行，大部分进程都被正确测试。

如图 5-2 可知，总样本数到达 500，错误率为 0.20%（仅 1 个请求失败），吞吐量达到 9.0 次/秒，平均响应时间为 553ms

5.2 详细测试结果

5.2.1 汇总报告数据

表 5-1 汇总报告

操作	样本数	平均响应时间(ms)	90%线(ms)	最小值(ms)	最大值(ms)	错误率	吞吐量(/s)
查询学生列表	100	**905**	1959	392	2800	0.00%	1.9
创建新学生	100	**900**	2050	423	2719	0.00%	1.9
查看学生详情	100	**7**	14	2	62	**1.00%**	1.9
编辑学生信息	100	**5**	12	1	90	0.00%	1.9

操作	样本数	平均响应时间(ms)	90%线(ms)	最小值(ms)	最大值(ms)	错误率	吞吐量(/s)
删除学生	100	**947**	1986	466	2682	0.00%	1.9
总计	**500**	**553**	**895**	**1**	**2800**	**0.20%**	**9.0**

5.3 性能指标深度分析

5.3.1 读写性能差异分析

从测试结果中观察到一个显著的性能差异现象：

1. 轻量级操作（编辑、详情）： 响应时间极快（平均 5-7ms）。这表明 EF Core 在通过主键（ID）进行点查询和更新非索引字段时，效率极高，且数据库缓存命中率良好。

2. 重量级操作（创建、删除、列表查询）： 响应时间较长（900ms+）。

5.3.2 稳定性与错误率

错误率分析： 整体错误率为 0.20%（1/500）。唯一的错误出现在"查看学生详情"中。

推测原因： 这是一个典型的并发竞争条件（Race Condition）。在多线程环境下，某个线程试图读取 ID 为 X 的学生详情时，另一个线程恰好执行了"删除 ID 为 X"的操作，导致查询因为找不到记录而抛出异常或返回 404。

5.3.3 吞吐量瓶颈

系统整体吞吐量为 9.0 TPS。对于本地环境而言，该数值可能受限于：

1. 数据库配置： SQL Server Express 版本的资源限制。

2. Web 服务器连接数： Kestrel 的默认并发连接限制。

3. 测试模式： 本次实验为了直观展示使用了 GUI 模式运行 JMeter，这消耗了大量客户端资源，实际生产测试应采用 CLI 命令行模式。

第 6 章 总结

6.1 实验结论

本次实验成功构建了基于 ASP.NET Core 的学生管理系统，并验证了其在高并发场景下的表现：

1. 功能完备性： 系统 100% 完成了预期的 CRUD 业务逻辑。

2. 性能特征： 系统呈现出"读极快、写较慢"的典型数据库应用特征。

3. 并发能力： 在 100 并发用户下，系统未发生崩溃，仅有极低概率的数据一致性错误，证明了系统架构的基本稳定性。

6.2 改进方向

1. 数据库优化： 针对 Index 页面涉及的查询字段添加索引；考虑将读写操作分离。

2. 并发控制： 引入乐观锁（Optimistic Locking）机制，在编辑和删除时检查 RowVersion，解决数据竞争导致的错误。

3. 测试优化： 后续应使用命令行模式（Non-GUI mode）运行 JMeter 以获得更准确的吞吐量数据。

参考文献

[1] Andrew Lock, ASP.NET Core in Action, Manning Publications, 2021

[2] Jon P Smith, Entity Framework Core in Action, Manning Publications, 2021