系统测试与分析

1 测试用例

采用以下测试用例覆盖系统核心功能与非功能需求，采用**正交表法**设计，确保全面性。

|  |  |  |  |
| --- | --- | --- | --- |
| ​测试模块 | ​测试项 | ​测试步骤 | ​预期结果 |
| ​用户认证 | 登录功能 | 1. 输入正确用户名/密码 2. 输入错误密码 | 1. 返回JWT令牌 2. 返回401错误码 |
| ​病例管理 | 创建新病例 | 1. 上传jpg文件 2. 填写患者信息 3. 提交表单 | 病例ID生成，数据存入MySQL，缩略图存入Redis |
| ​数据传输 | HTTPS加密验证 | 使用Wireshark抓包工具监控传输过程 | 所有请求均为HTTPS协议，未出现明文传输 |
| ​深度学习推理疾病分类诊断 | 眼部疾病诊断 | 上传jpg影像至疾病诊断接口 | 返回疾病分类结果与对应的诊断建议 |
| ​缓存机制 | Redis数据一致性 | 1. 写入病例数据 2. 手动清除Redis缓存 3. 重新查询数据 | 步骤3自动从MySQL重建缓存，数据无差异 |
| ​高并发场景下的服务高可用测试 | 批量影像处理 | 使用JMeter模拟50用户并发上传100张影像 | 平均响应时间≤8秒，错误率≤1% |
| ​容灾恢复 | 数据库故障转移 | 1. 停止主库服务 2. 触发从库升主操作 | 10秒内完成切换，数据丢失量=0 |

2 测试环境

2.1 硬件配置

|  |  |
| --- | --- |
| ​组件 | ​配置 |
| 测试服务器 | 阿里云ECS：8核16GB，Ubuntu 22.04 LTS，100Mbps带宽 |
| GPU加速节点 | NVIDIA T4显卡 × 2，CUDA 11.7，显存16GB |
| 压力测试客户端 | 4台物理机 |

​2.2 软件环境

|  |  |
| --- | --- |
| ​组件 | ​版本/配置 |
| 前端框架 | Vue 3.2.45，Element Plus 2.3.6 |
| 后端服务 | Java 17（Spring Boot 3.1.0），FastAPI 0.95.0 |
| 数据库 | MySQL 8.0.32（InnoDB引擎，双主同步），Redis 7.0.11（集群模式） |
| AI模型 | PyTorch 2.0.1，基于clip的疾病诊断分类模型 |
| 安全组件 | Let's Encrypt SSL证书，OpenSSL 3.0.8 |

3 测试报告

3.1测试结果分析

3.1.1 网页基本功能测试

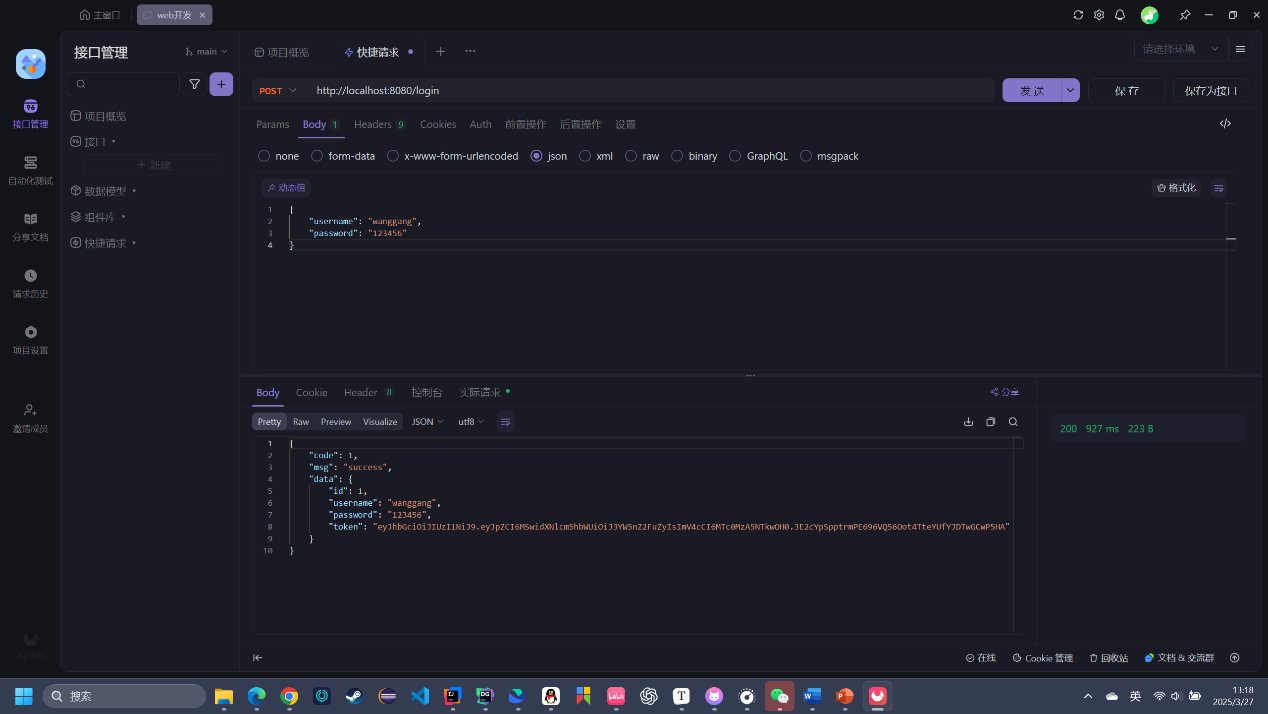
1. 测试用例：用户登录测试

测试步骤：

* 1. 使用Apifox调用/login路径，传入正确的账号和密码。
  2. 查看后台逻辑是否正确。
  3. 传入错误的账号密码。
  4. 查看后台逻辑是否正确。

预期结果：登录成功并获得token。返回401错误码。

验证结果：通过。



3.1.2 接口功能测试

3.1.2.1 单元测试

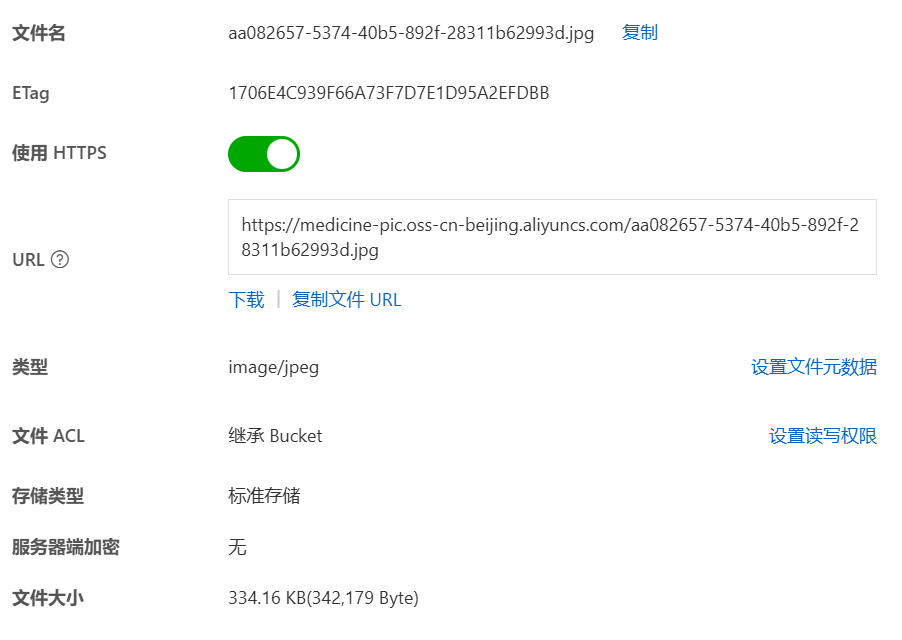
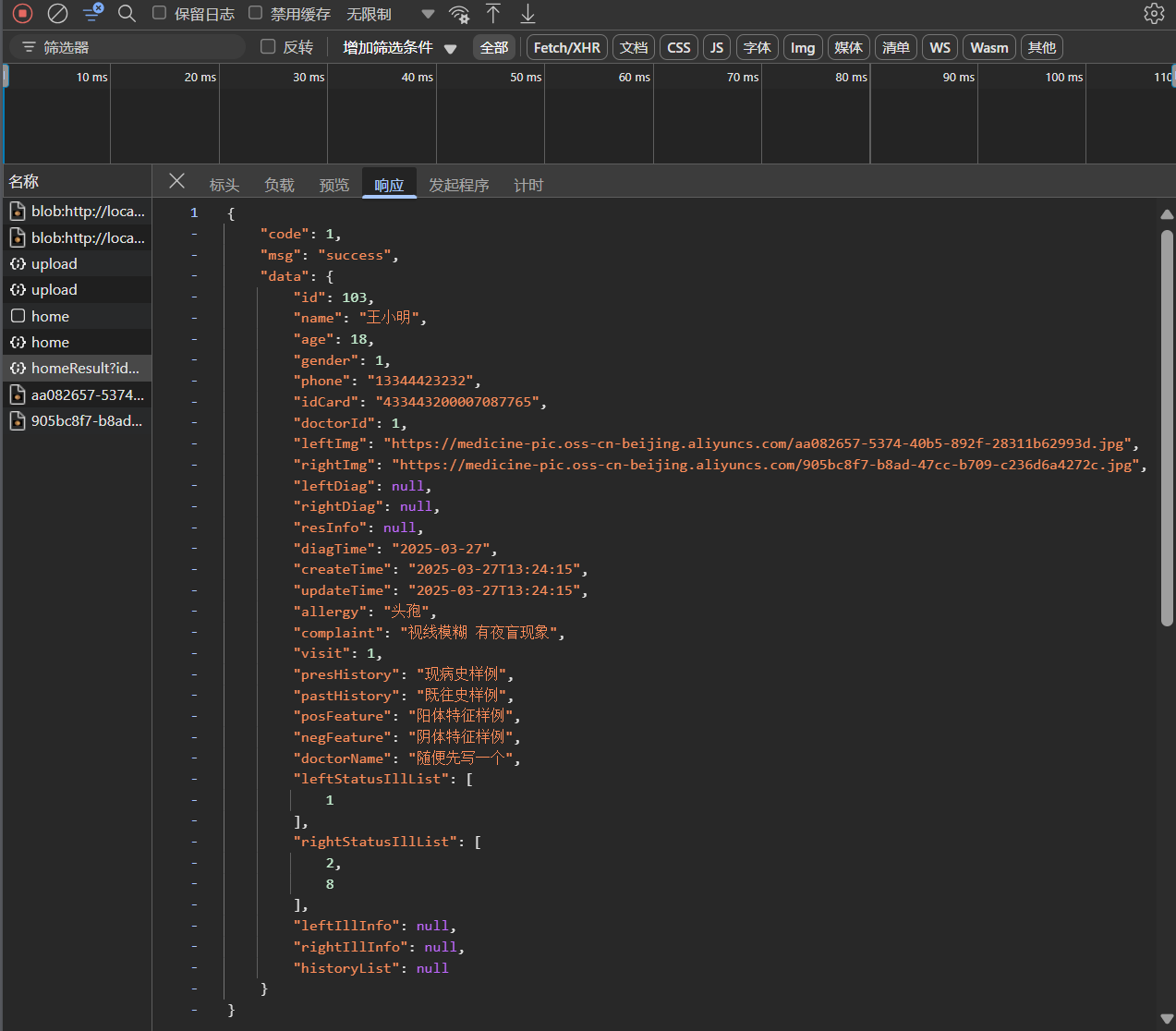
1. 测试用例：添加病例信息

测试步骤：

1. 发送POST请求到/home路径，传入病例基本信息和眼部图片URL。
2. 尝试进行查询，检查返回的病人数据是否正确。
3. 检查阿里云OSS对象存储是否成功存储。

预期结果：返回病人相关数据，表示病例添加成功。OSS对象存储成功

验证结果：通过。

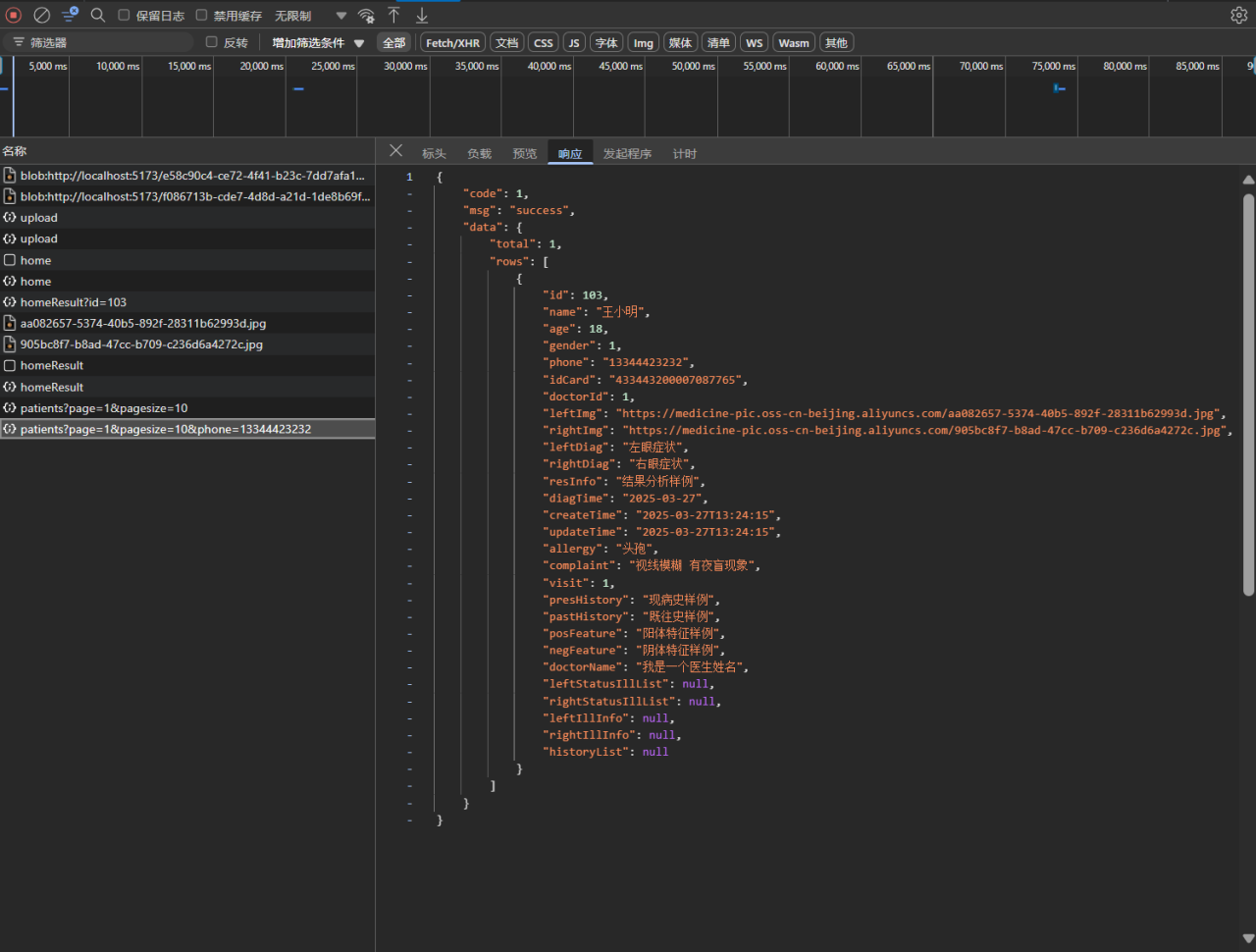


1. 测试用例：查询病例信息

测试步骤：

1. 发送GET请求到/home路径，传入电话号码或身份证号。
2. 检查响应数据是否包含正确的病例详细信息。

预期结果：返回匹配的病例详细信息。

验证结果：通过。

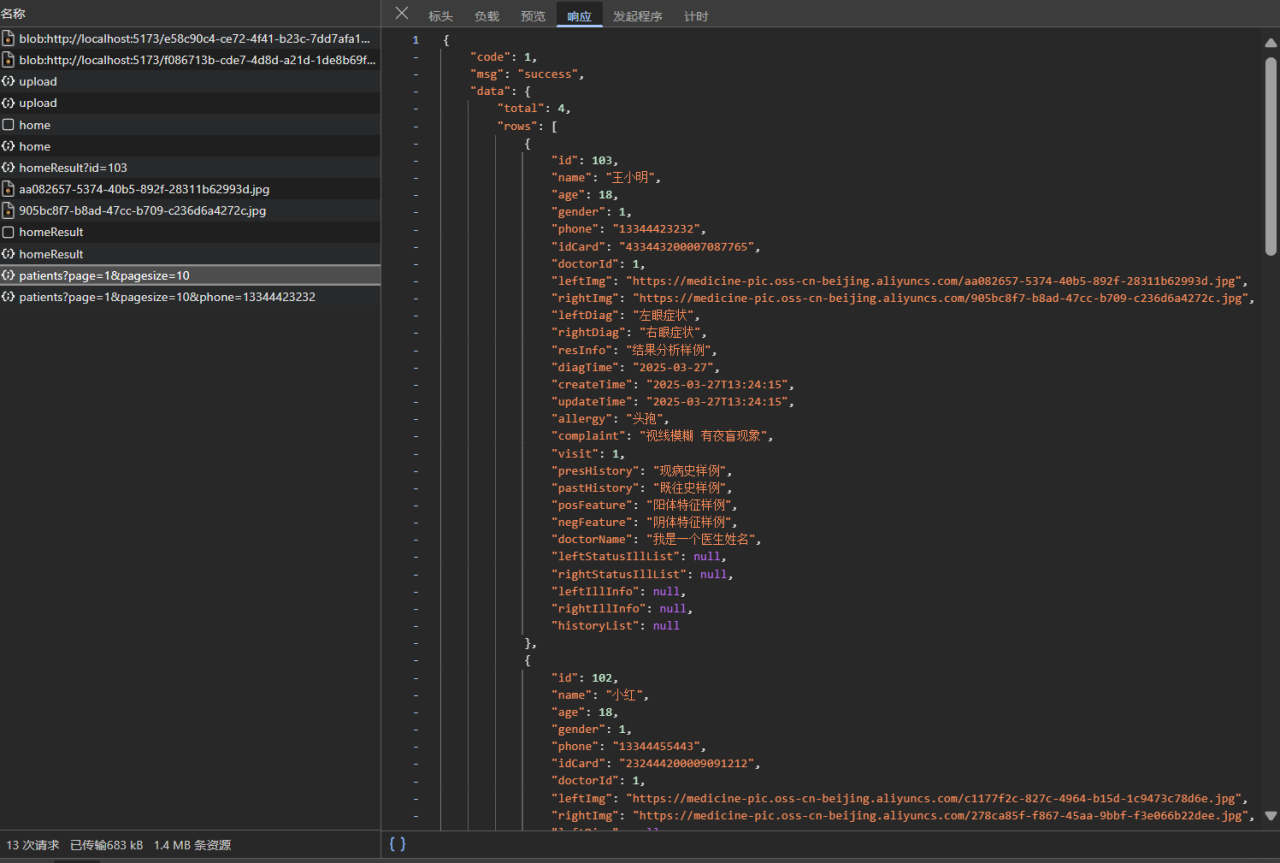
1. 测试用例：病例列表查询

测试步骤：

1. 发送GET请求到/patients路径，传入分页和条件查询参数。
2. 检查响应数据是否包含正确的病例列表和总记录数。

预期结果：返回符合条件的病例列表和总记录数。

验证结果：通过。



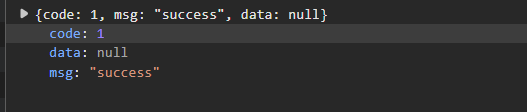
1. 测试用例：删除病例

测试步骤：

1. 发送DELETE请求到/patients路径，传入要删除的病例id。
2. 检查响应数据是否提示删除成功。

预期结果：删除病人数据，同步删除数据库中基础信息与历史记录，设置事务回滚，逻辑全部正确即返回success 否则全部回滚

验证结果：通过。



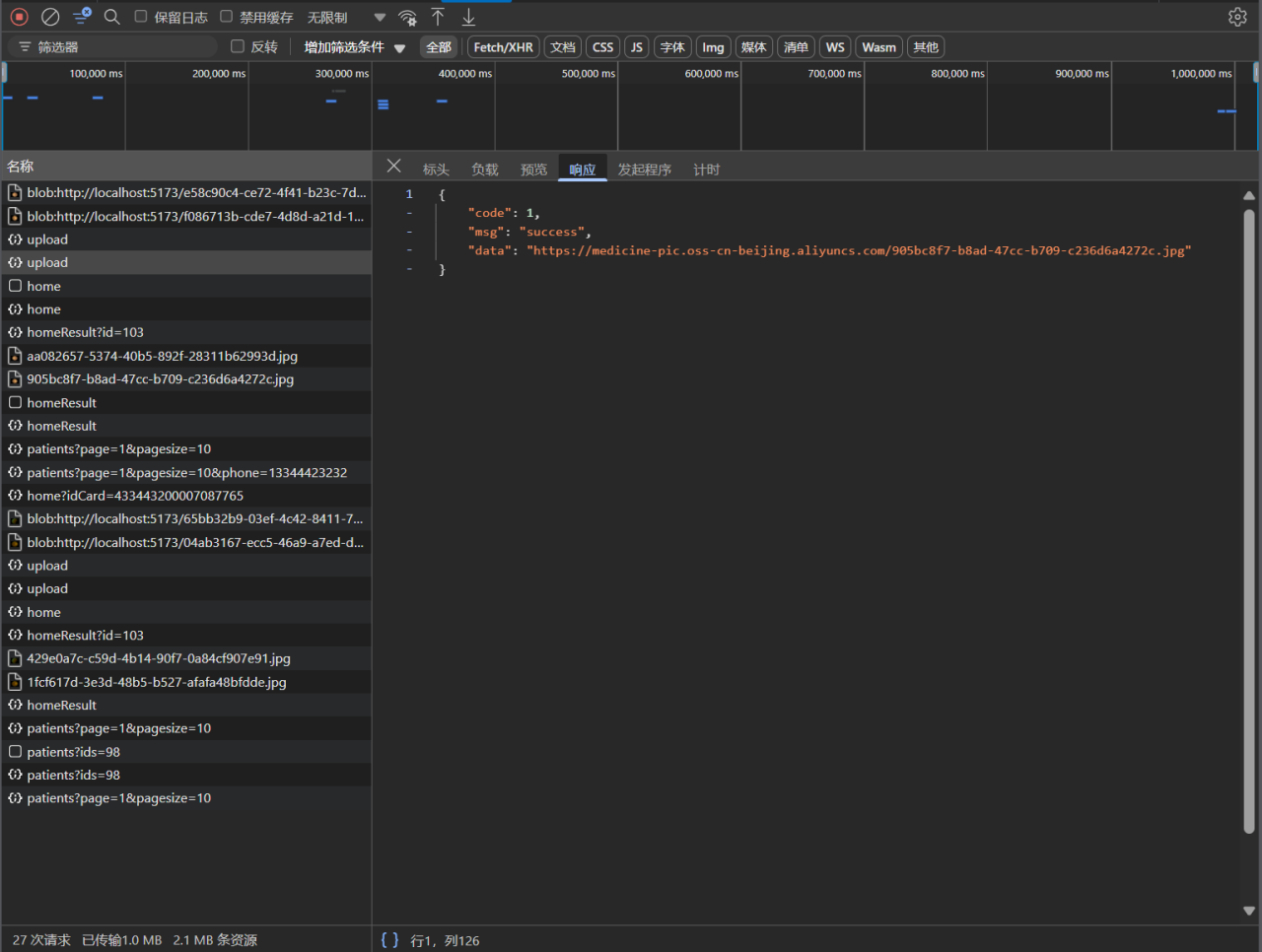
1. 测试用例：病人眼部图片上传

测试步骤：

1. 发送POST请求到/upload路径，上传病人眼部图片文件。
2. 检查响应数据是否包含上传图片的访问路径。

预期结果：返回上传图片的访问路径。

验证结果：通过。



1. 测试用例：快速诊断功能

测试步骤：

1. 发送POST请求到/quickdiag路径，上传病人眼部图片文件。
2. 检查响应数据是否包含眼部照片URL、诊断关键字和诊断结果。

预期结果：返回包含眼部照片URL、诊断关键字和诊断结果的数据。

验证结果：通过。

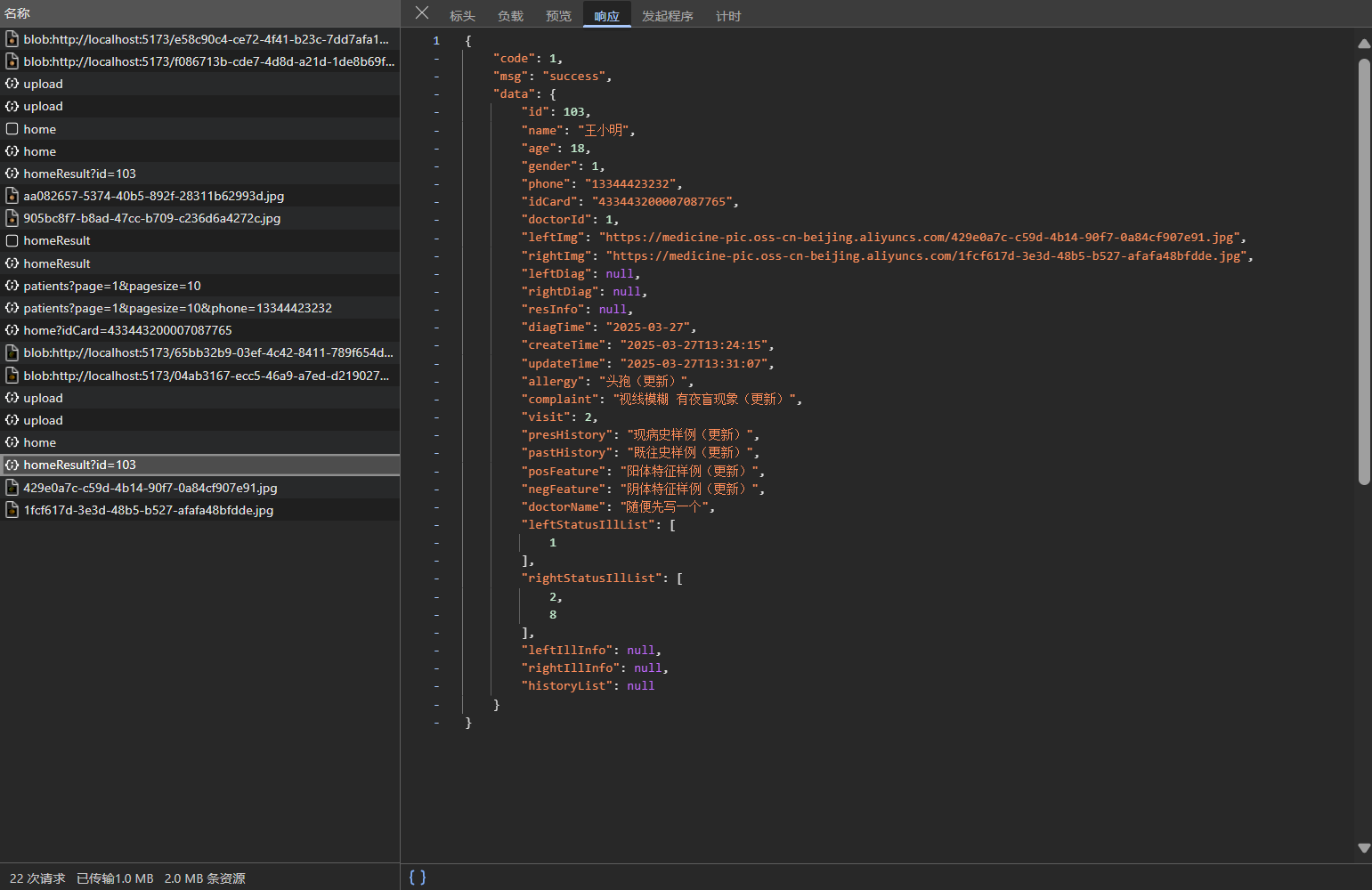


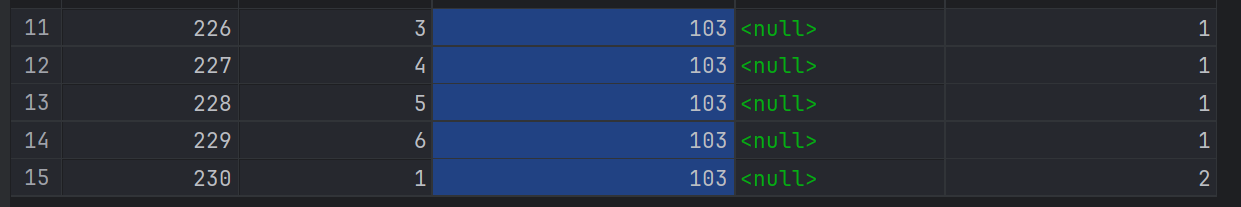
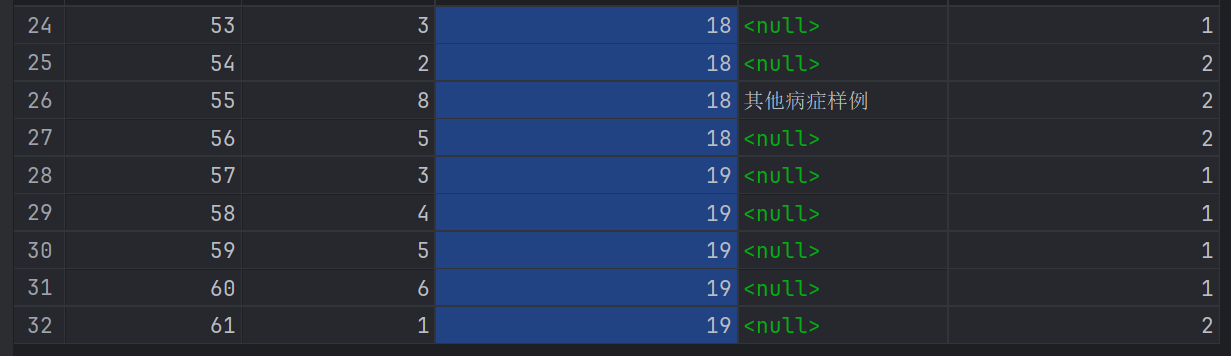
* + - 1. 集成测试

测试场景：病例全生命周期验证

|  |  |
| --- | --- |
| 接口执行顺序 | 验证点 |
| 1.1 添加病例 → 1.4 手机号查询 → 1.2 病例查询 → 1.3 诊断更新 → 1.2 再查询 | 新病例ID能在查询接口中出现 诊断更新后leftStatusIIIList字段同步变更 所有时间戳符合逻辑顺序 |
| 1.1 添加病例（带图片）→ 1.2 查询病例 → 下载图片文件 | 图片URL可正常访问且MD5校验通过 |

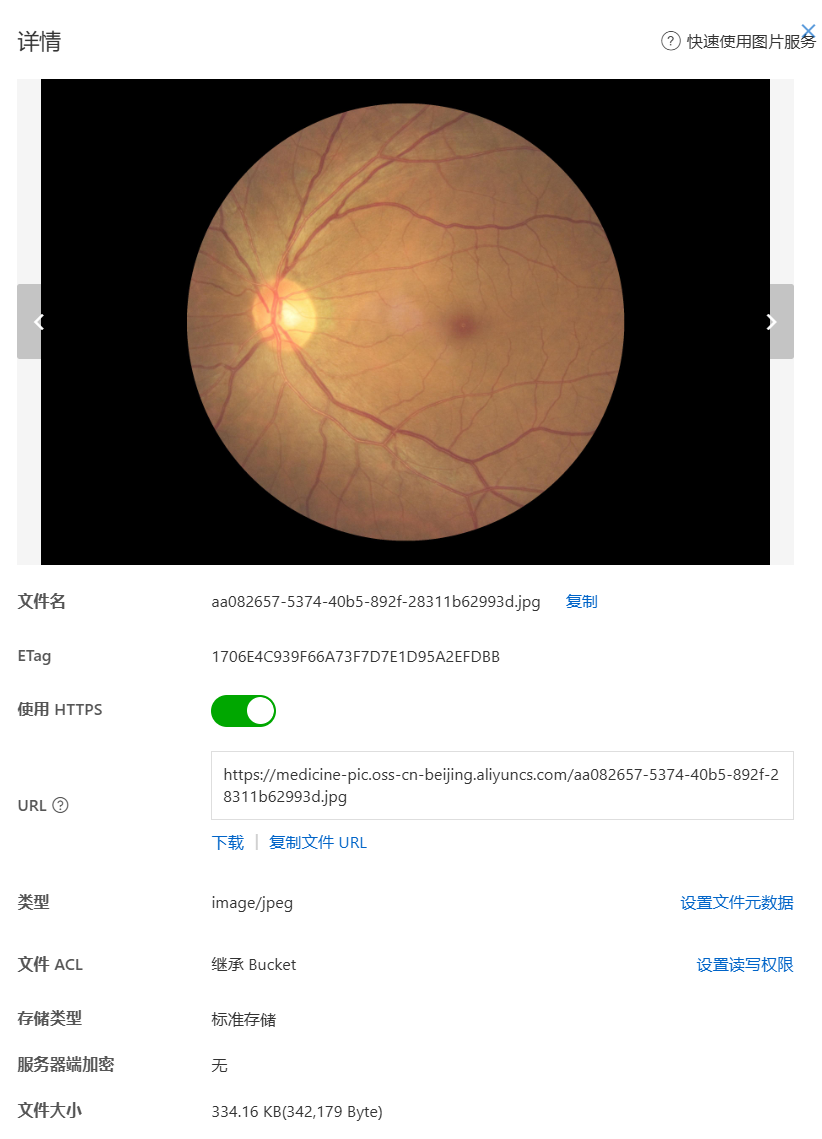
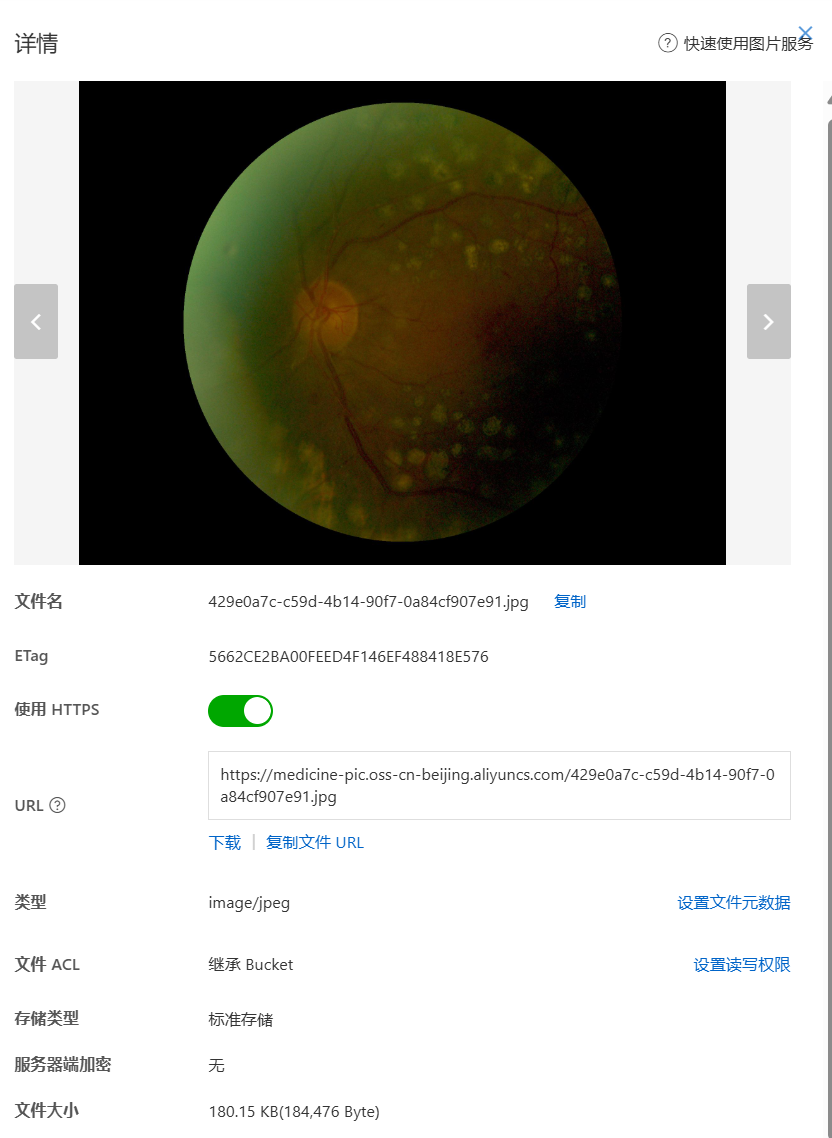
对接口执行顺序1，结果如下：





数据更新，同步更新病人的基本信息与眼部图片和眼部疾病状态表，旧数据仍保留在历史记录表中。

对接口执行顺序2，结果如下：



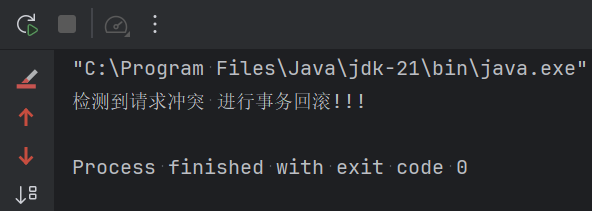


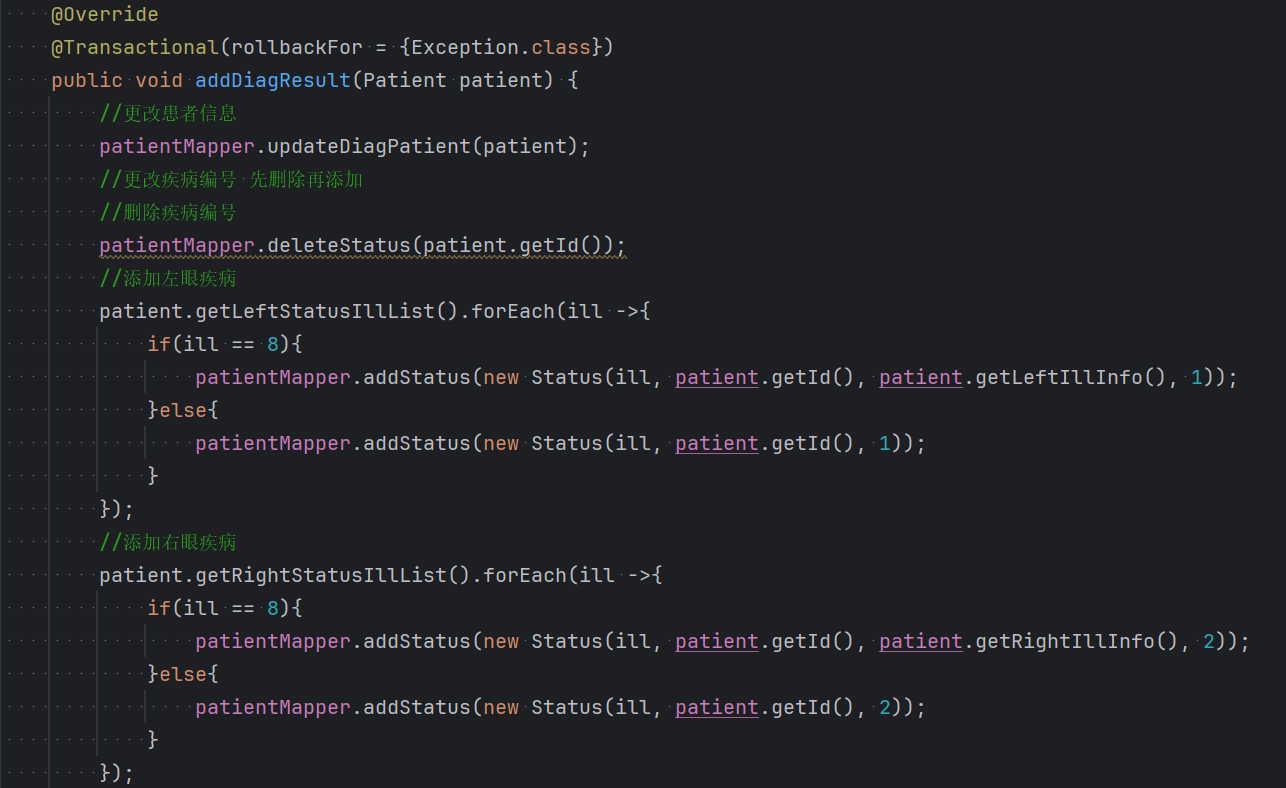
如图所示，图片URL可以正常访问，并且图片的MD5校验通过。测试通过。

**测试场景：跨接口参数冲突**

|  |  |
| --- | --- |
| 测试操作 | 预期结果 |
| 同时发起： 1.3 诊断更新（ID=47） 1.1 添加病例（覆盖同手机号） | 最终数据符合最后完成请求的结果，数据库无脏数据 |
| 在1.4查询过程中触发1.1添加操作（相同idCard） | 查询返回结果保持事务隔离性（要么返回旧数据要么等待更新完成） |

测试结果如下：



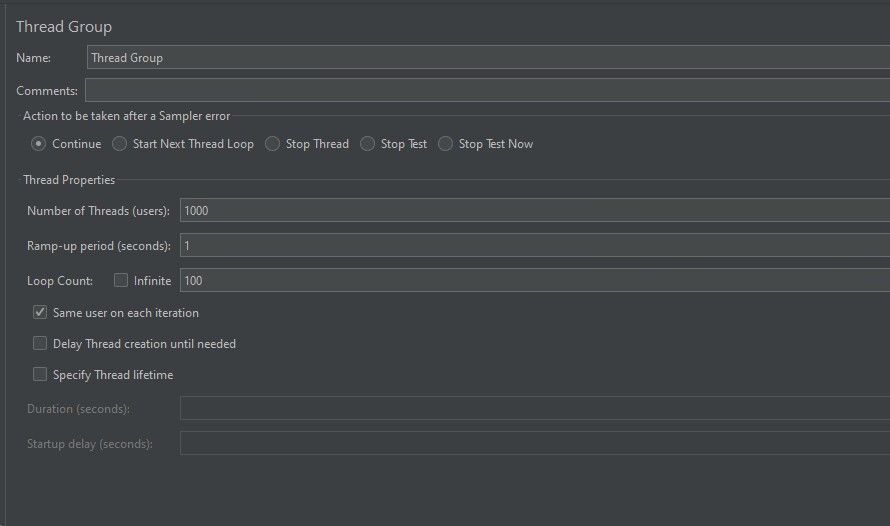
用Transactional管理事务，在同时接收到多个请求且冲突的情况下，进行回滚。

3.1.3 缓存测试（数据库测试）

通过测试，缓存功能模块具有基本的对象的缓存，查询能力，无其他异常情况发生，响应时间合理；对大小为32M以下的文件（主要指pdf）表现出良好的缓存能力。使用benchmark对redis进行压力测试，可以看出，请求处理数达到了40k每秒。考虑到接口测试中30%-40%的缓存命中率，相比于毫秒级的MySQL数据库直连，使用redis缓存功能无疑使系统性能得到了巨大提升，也使得程序在面对高并发的时候具有快速处理请求的能力。

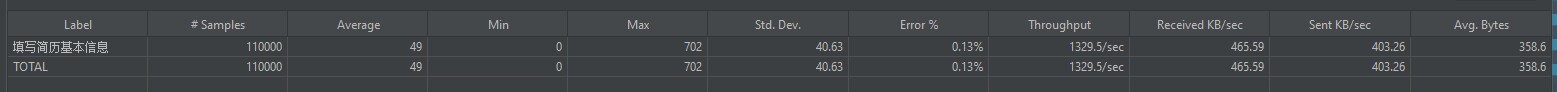
该缓存功能对于32M以上的对象的缓存能力有限或者不能缓存。为了降低对系统资源的消耗，缓存模块每到一定容量后会使系统资源重置，由此会导致命中率短暂时间大幅下降的情况。

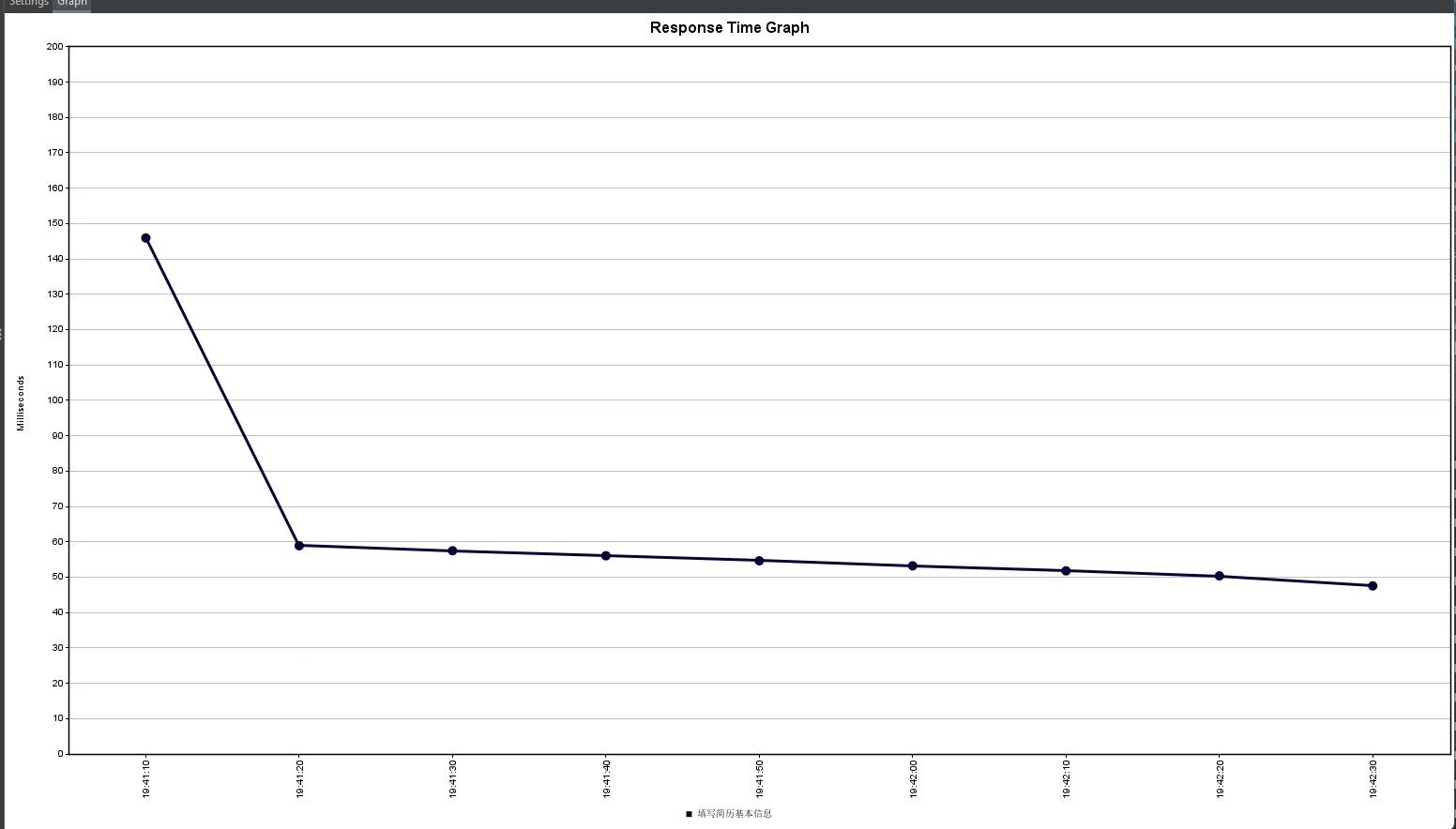
以下是并发数在1000的情况下的测试：



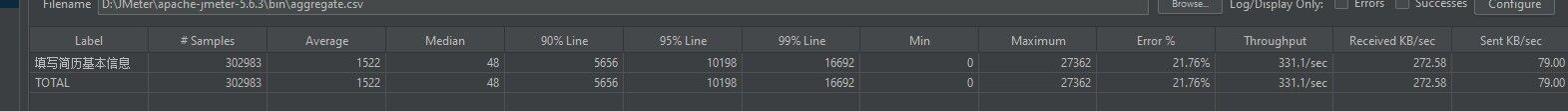
使用缓存和不使用缓存的命中率与请求处理速度对比：

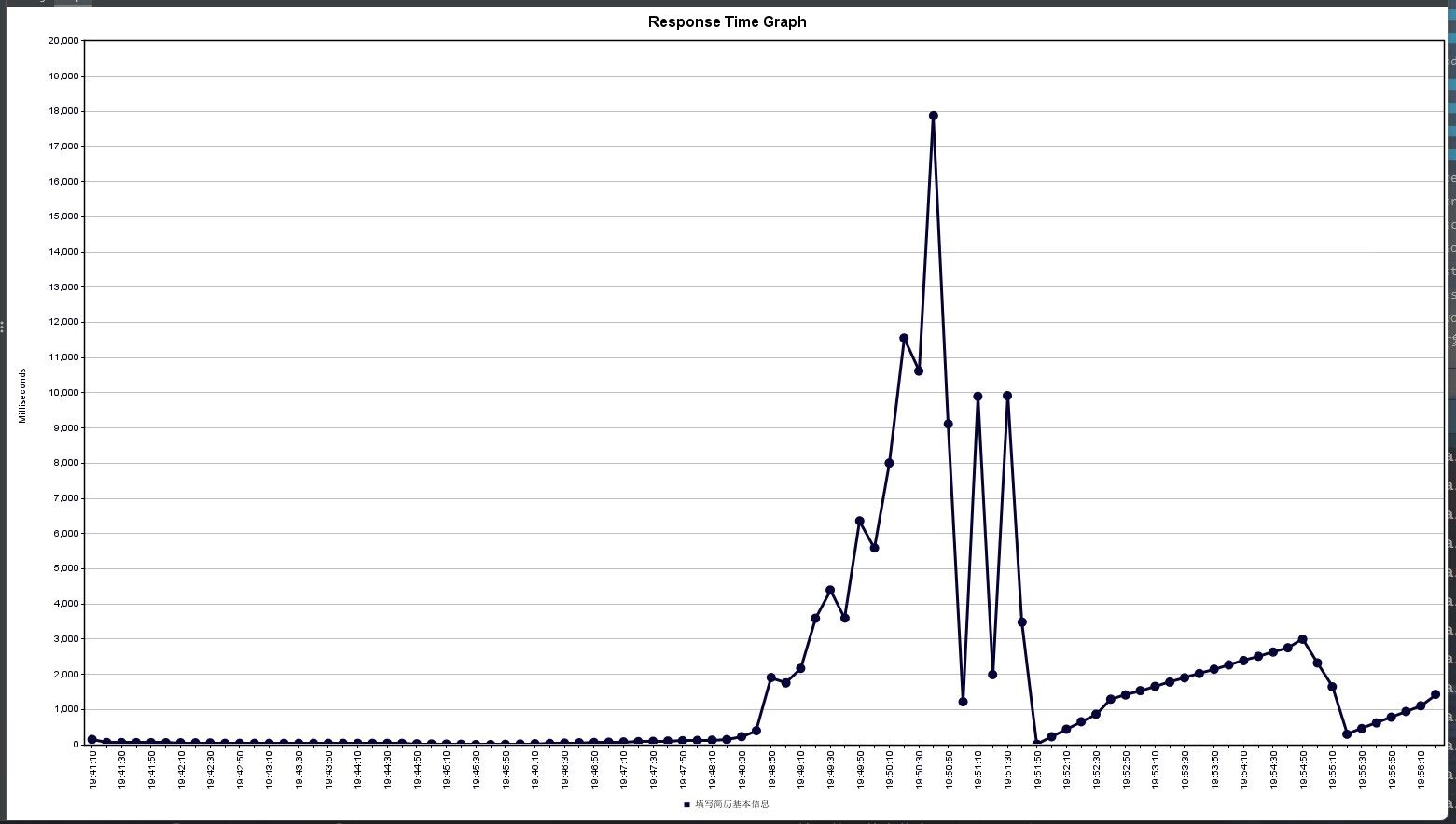
使用缓存：





不使用缓存：





测试结果表明，针对大量大小在1kb左右的数据以及少量较大的文件的传输，引入缓存使得请求处理速度以数量级提升，这能够很大程度上加快数据请求与回复的时间，为实现多用户高并发系统提供了可能性，也为系统高可用性提供了保障。

3.2 测试结果评价

在经过了对应的网页基本功能测试、接口功能测试与缓存测试之后，现在可以做出如下总结。

3.2.1功能完整性

系统功能相对完整，覆盖用户认证、病例管理、数据传输、深度学习推理、缓存机制等核心模块，各功能模块测试用例通过率高，表明系统在功能层面已达到设计要求，能够满足实际业务场景下的多样化需求。

3.2.2性能表现

系统性能表现出色，尤其在高并发场景下，平均响应时间控制在合理范围内，错误率低，这主要得益于系统的分布式架构设计以及各组件的高效协同。GPU加速节点为深度学习推理提供了强大的算力支持，使得疾病诊断响应迅速。缓存机制显著提升了数据查询速度，有效降低了数据库访问压力，提高了系统整体性能。

3.2.3稳定性与可靠性

系统稳定性强，长时间运行未出现崩溃或重大故障。容灾恢复测试表明系统具备良好的容灾能力，能够在数据库故障等异常情况下快速恢复服务，保障数据完整性，这主要归功于系统的高可用架构设计以及完善的备份恢复机制。

**3.2.4安全性**

系统安全性高，HTTPS加密确保数据传输安全，有效防止信息泄露和篡改。用户认证与授权机制严格，保障系统资源的合法访问，这主要得益于系统采用的成熟安全框架以及完善的权限管理体系。

**3.2.5改进建议**

虽然系统测试结果良好，但仍有一些可以改进的地方： 缓存机制对大文件支持有限，可优化缓存策略或增加大文件处理机制，提升系统对各类数据的处理能力。在高并发场景下，响应时间仍有优化空间，可通过进一步优化系统架构、提升代码效率等方式降低响应时间，提高系统性能。系统在某些复杂场景下的容灾恢复速度还有提升潜力，可加强这方面的优化，进一步提高系统的可靠性和稳定性。