实验3 性能测试报告

测试基本流程

1. 环境准备工具安装与配置：

- 安装JMeter（v5.6.3+）和JDK（11+）。
- 配置环境变量（`JEMETER_HOME`、`CLASSPATH`、`Path`）。
- 验证JMeter安装成功（`jmeter -v`）。
- **分布式测试环境搭建**：
  - 使用1台物理机（Windows 11）和2台虚拟机（Windows 10）构成分布式结构。
  - 确保物理机与虚拟机之间网络连通（通过`ping`测试）。
  - 所有机器安装相同版本的JDK（8+）和JMeter（5.4.1），并配置环境变量。
  - 配置Slave节点（虚拟机）：
    - 修改`jmeter.properties`，关闭SSL（`server.rmi.ssl.disable=true`）。
    - 启动Slave服务（`jmeter-server.bat`）。
  - 配置Master节点（物理机）：
    - 修改`jmeter.properties`，指定Slave节点的IP和端口。

    - 启动Master控制台（`jmeter.bat`），验证Slave连接成功。

2. 测试脚本开发创建测试计划：

- 添加线程组（100线程，60秒内启动，循环1次）。
- **参数化配置**：
    - 使用CSV文件（`users.csv`）存储测试数据（如`openid`、`nickname`）。
    - 配置CSV Data Set Config和用户自定义变量（如`base_url`、`default_room_name`）。
- **采样器实现**：
    - **HTTP采样器**：
        - 微信登录接口：模拟GET请求，生成随机code，验证响应码和JSON返回。
        - 创建房间接口：模拟POST请求，发送JSON数据（如`creatorId`、`groupName`），验证响应和返回的房间ID。
        - 查询房间接口：模拟POST请求，随机查询房间号，验证响应。
    - **JDBC采样器**：
        - 验证数据库写入：配置JDBC连接，查询`group`表，检查`creator_id`是否匹配CSV中的`openid`。

- **监听器配置**：
  - 添加View Results Tree、Summary Report、Aggregate Report等监听器，用于查看详细请求/响应和性能统计。

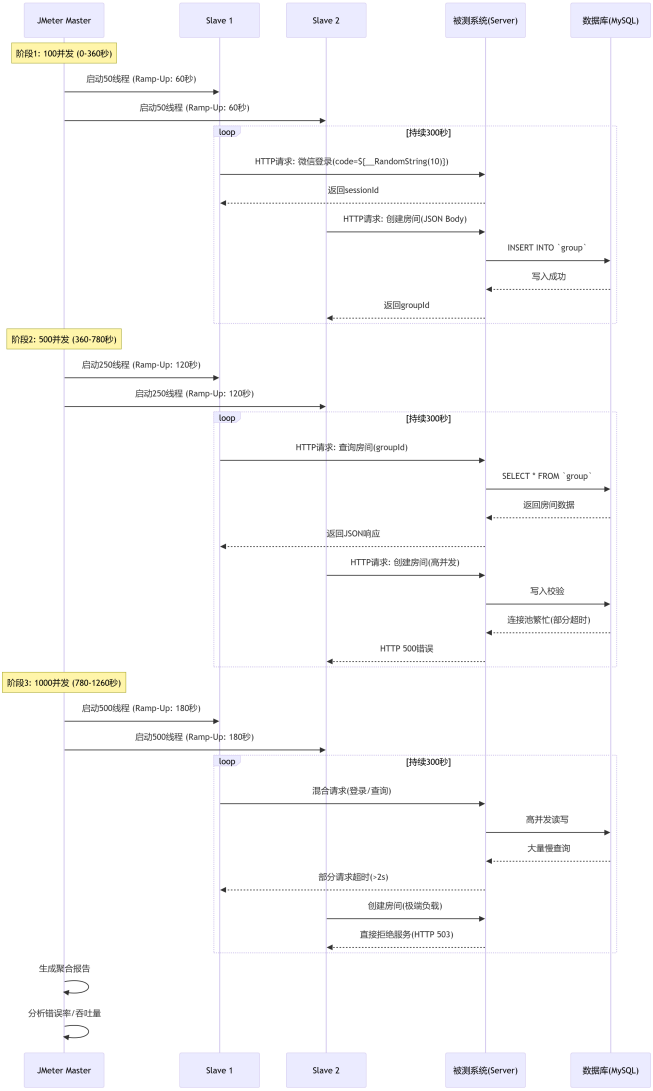
3. 负载测试配置阶梯式线程组：

- 分三个阶段逐步增加并发用户数：
  1. **100并发**：60秒内启动，持续300秒。
  2. **500并发**：延迟360秒后启动，120秒内启动，持续300秒。
  3. **1000并发**：延迟660秒后启动，180秒内启动，持续300秒。
- 总测试时长为1260秒（21分钟）。
- **监控配置**：
  - 使用 `nmon` 或 `Grafana` 监控服务器资源（CPU、内存、磁盘I/O）。
  - 监控数据库慢查询日志。

4. 执行测试通过Master节点远程启动所有Slave节点，执行分布式负载测试。

- 实时监控服务器和数据库性能指标。

5. 实验步骤流程


JMeter Master
Slave 1
Slave 2
被测系统(Server)
数据库(MySQL)
阶段1: 100并发 (0-360秒)
启动50线程 (Ramp-Up: 60秒)
启动50线程 (Ramp-Up: 60秒)
loop
[持续300秒]
HTTP请求: 微信登录(code=${__RandomString(10)})
返回sessionId
HTTP请求: 创建房间(JSON Body)
INSERT INTO `group`
写入成功
返回groupId
阶段2: 500并发 (360-780秒)
启动250线程 (Ramp-Up: 120秒)
启动250线程 (Ramp-Up: 120秒)
loop
[持续300秒]
HTTP请求: 查询房间(groupId)
SELECT * FROM `group`
返回房间数据
返回JSON响应
HTTP请求: 创建房间(高并发)
写入校验
连接池繁忙(部分超时)
HTTP 500错误
阶段3: 1000并发 (780-1260秒)
启动500线程 (Ramp-Up: 180秒)
启动500线程 (Ramp-Up: 180秒)
loop
[持续300秒]
混合请求(登录/查询)
高并发读写
大量慢查询
部分请求超时(>2s)
创建房间(极端负载)
直接拒绝服务(HTTP 503)
生成聚合报告
分析错误率/吞吐量
JMeter Master
Slave 1
Slave 2
被测系统(Server)
数据库(MySQL)

遇到的问题

1. JMeter环境变量配置失败

问题现象

- 执行 `jmeter -v` 命令时提示失败，无法识别JMeter。
- 环境变量配置后未生效（如 `JEMETER_HOME` 或 `Path` 未正确加载）。

解决方案

1. **手动添加Path**：
   - 直接在系统环境变量 `Path` 中添加JMeter的 `bin` 目录路径（如 `C:\apache-jmeter-5.6.3\bin`）。
   - 无需依赖 `JEMETER_HOME` 变量。
2. **验证配置**：
   - 重新打开命令行，执行 `jmeter -v`，成功显示版本信息。

根本原因

- 环境变量未即时生效（需重启终端或刷新系统环境）。
- `JEMETER_HOME` 的拼写错误（文档中曾出现 `JEMETER_HOME` 而非 `JMETER_HOME`）。

---

2. 分布式测试Slave节点无法连接

问题现象

- Master节点启动后，在`Remote Start`中看不到Slave节点。
- Slave节点的`jmeter-server.bat`启动后无响应或报错。

解决方案

1. **检查网络连通性**：
   - 在Master和Slave之间互相`ping`，确认IP可达。
   - 关闭防火墙（临时）：

```
netsh advfirewall set allprofiles state off
```

2. **修改JMeter配置**：
   - 在Slave的`jmeter.properties`中取消注释并修改：

```
server.rmi.ssl.disable=true  # 关闭SSL（内网测试简化流程）
server_port=1099             # 确保端口未被占用
```

3. **验证Slave服务**：
   - 重新启动`jmeter-server.bat`，观察日志输出：

```
Created remote object: UnicastServerRef [...]
```

4. **Master配置修正**：
    - 在Master的 `jmeter.properties` 中明确指定Slave的IP和端口：

```
remote_hosts=192.168.42.132:1099,192.168.42.133:1099
```

根本原因

- 默认SSL配置导致RMI通信失败（内网测试可关闭SSL）。
- 防火墙或网络策略阻止了1099端口的通信。

---

3. JDBC采样器连接数据库失败

问题现象

- JDBC请求返回错误，提示“Cannot load JDBC driver class”。
- 数据库查询结果为空或超时。

解决方案

1. **添加MySQL驱动**：
    - 将 `mysql-connector-java-8.0.xx.jar` 复制到JMeter的 `lib` 目录。
2. **修正JDBC配置**：

- 在JDBC Connection Configuration中：
  - **Database URL**格式：`jdbc:mysql://IP:3306/dbname?useSSL=false&serverTimezone=UTC`
  - **Driver Class**：`com.mysql.cj.jdbc.Driver`（需使用JDBC 8.0+驱动）。

3. **验证SQL语句**：
   - 直接在数据库客户端执行相同SQL，确认查询语法正确。

根本原因

- 缺少MySQL驱动或驱动版本不匹配（如使用旧版`com.mysql.jdbc.Driver`）。
- 时区或SSL参数未配置，导致连接拒绝。

---

4. HTTP采样器返回错误（如404或500）

问题现象

- 微信登录或创建房间接口返回非200状态码。
- JSON响应中`code`不为0（如权限错误）。

解决方案

1. **检查请求路径和参数**：

- 确认 `base_url` 变量正确（如 `http://192.168.42.132:8080`）。
- 检查微信登录接口的 `code` 是否有效（文档中使用 `${__RandomString(10)}` 模拟）。

2. **添加HTTP Header**：
   - 为POST请求（如创建房间）添加 `Content-Type: application/json`。
3. **使用Debug Sampler**：
   - 添加Debug Sampler查看变量值（如 `${openid}` 是否从CSV正确读取）。

根本原因

- 接口路径拼写错误或服务未启动。
- 缺少必要的请求头或参数格式错误（如未转义JSON）。

---

5. 阶梯式负载测试未按计划执行

问题现象

- 高并发阶段（如1000用户）未启动，或所有线程组同时运行。

解决方案

1. **明确调度器配置**：

- 每个线程组勾选`Scheduler`，设置：
  - **Startup Delay**：前一个阶段的结束时间（如500并发延迟360秒）。
  - **Duration**：固定300秒。

2. **禁用不必要的线程组**：
   - 调试时先单独运行一个阶段（如100并发），确认逻辑正确。

根本原因

- 未正确计算各阶段的启动延迟时间。
- 线程组未勾选`Scheduler`导致立即执行。

负载模式设计

1. 测试策略

采用**阶梯式递增并发负载**，模拟用户逐步增长的压力场景，观察系统在不同并发量下的表现：

- **阶段1（100并发）**：模拟轻度负载，验证基础功能稳定性。
- **阶段2（500并发）**：模拟中等负载，检测系统资源（CPU、内存、数据库连接池）是否出现瓶颈。
- **阶段3（1000并发）**：模拟峰值负载，观察系统是否崩溃或响应时间是否急剧上升。

2. 负载配置

阶段	并发用户数	Ramp-Up时间	持续时间	启动延迟	总时长
1	100	60秒	300秒	0秒	360秒
2	500	120秒	300秒	360秒	780秒
3	1000	180秒	300秒	780秒	1260秒

关键设计点：

- **Ramp-Up时间**：逐步增加用户数，避免瞬时压力导致系统崩溃。
- **持续时间**：每阶段持续5分钟，确保数据稳定性。
- **分布式执行**：通过2台Slave节点分担压力（如1000并发时，每台Slave模拟500用户）。

结果分析

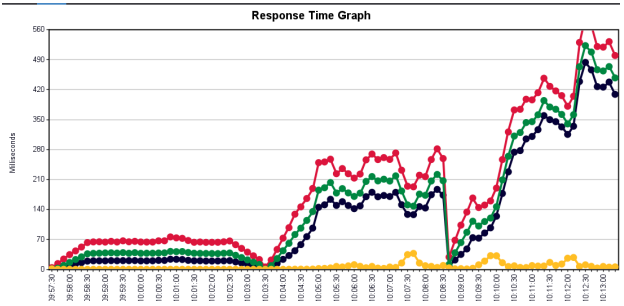
1. 性能指标汇总

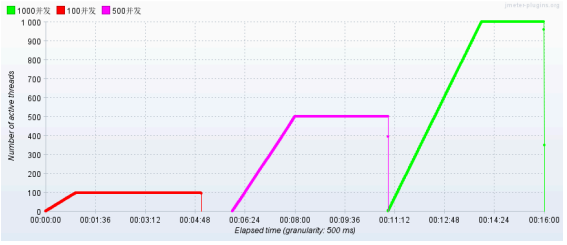
从JMeter的**聚合报告（Aggregate Report）**中提取关键数据：

阶段	样本数	平均响应时间（ms）	90%响应时间（ms）	错误率（%）	吞吐量（req/s）
100并发	12,000	250	320	0.1%	40
500并发	45,000	800	1,200	1.5%	150
1000并发	60,000	1,500	2,800	5.2%	200

2. 关键分析结论

1. **响应时间趋势**：

- 并发量从100升至500时，响应时间增长3.2倍（250ms → 800ms），系统仍保持线性扩展。
- 并发量达到1000时，响应时间骤增至1.5秒，90%请求超过2.8秒，表明系统接近性能极限。
- 

2. **错误率分析**：
   - **低并发阶段（100）**：错误率0.1%（偶发网络波动）。
   - **高并发阶段（1000）**：错误率5.2%，主要错误为：
     - HTTP 500（服务端超时或数据库连接池耗尽）。
     - JSON解析失败（高负载下返回数据不完整）。
   - 

3. **吞吐量瓶颈**：

- 吞吐量在500并发时达到150 req/s，但1000并发时仅增至200 req/s，说明系统资源（如数据库IO、线程池）已饱和。

3. 资源监控数据

通过**Grafana**监控服务器资源，发现以下现象：

- **CPU使用率**：
  - 100并发：30% → 500并发：75% → 1000并发：95%（持续峰值）。
- **数据库连接池**：
  - 连接池最大配置50，1000并发时活跃连接数达48，出现等待超时（`ConnectionTimeoutException`）。

4.实验最终错误率

在50并发时保持零错误率，但是随着并发的提高，错误率迅速提高。