分布式系统实验报告

实验一

RMI分布式通信及其应用

姓名： 常添

学号： 2022111699

院系： 计算学部

2025年 5 月 19 日

**开发环境：JDK 17 + IntelliJ IDEA 2024 + Windows 11**

**一、实验目的**

1. 掌握 Java RMI（Remote Method Invocation）机制的基本概念与工作流程。
2. 能够设计并实现一个包含客户端 / 服务器 / 远程接口的分布式应用。
3. 练习并发控制与错误处理：确保多用户同时访问共享资源时数据一致、系统稳定。
4. 通过自动化脚本 验证远程调用正确性、并发排他性及容错能力。

**二、原理概述**

| 关键概念 | 作用 | 本实验体现 |
| --- | --- | --- |
| 远程接口 (Remote Interface) | 描述可跨 JVM 调用的方法 | common.LibraryService 继承 Remote |
| 远程对象 (Remote Object) | 接口具体实现，由服务器导出 | server.LibraryServiceImpl 继承 UnicastRemoteObject |
| Stub / Skeleton | 客户端代理 / 服务器适配层，负责序列化与网络通信 | JDK 17 运行期自动生成，无需 rmic |
| RMI Registry | 名称服务，维护“字符串 → 远程引用”映射 | 嵌入式启动：LocateRegistry.createRegistry(1099) |
| 分布式 GC | Stub 与服务器保持租约，租约到期即回收远程对象 | JDK 内部完成，无需额外编码 |

**一次远程调用流程：**  
客户端 Stub → 序列化方法与参数 → TCP 发送 → 服务器 Skeleton → 反射执行真实方法 → 返回值序列化 → Stub 反序列化 → 调用者获得结果。

**三、系统设计**

**3.1 功能需求**

| 角色 | 功能 |
| --- | --- |
| 用户（客户端） | ① 查看图书列表 ② 借阅图书 ③ 归还图书 |
| 服务器 | ① 维护图书状态（在库 / 借出 / 借阅人） ② 处理并发请求并保证一致性 ③ 返回友好的业务提示 |

**3.2 模块划分与类图**

common

├── Book // 可序列化实体

└── LibraryService // 远程接口

server

├── LibraryServiceImpl // 远程对象实现

└── LibraryServer // 启动 RMI + 注册

client

└── LibraryClient // 交互式 CLI

test

└── LibrarySystemTest // 自动验收脚本

Book 内部字段 borrower 和操作方法 borrow()/giveBack() 用 synchronized 修饰；LibraryServiceImpl 的公共方法同样 synchronized，确保在多线程、跨客户端场景下的原子性。

**3.3 数据结构**

| 名称 | 类型 | 说明 |
| --- | --- | --- |
| books | HashMap<String, Book> | key=书名，value=Book 对象；存于服务器内存 |
| bindings(Registry) | ConcurrentHashMap<String, Remote> | JDK 内部使用；存储服务名与远程引用 |

**四、核心实现**

| 位置 | 关键实现要点 |
| --- | --- |
| Book.java | ① 实现 Serializable；② synchronized 包住状态变更；③ toString() 动态展示借阅人 |
| LibraryServiceImpl.java | ① 在构造器初始化 3 本示例书；② 所有公有方法 synchronized；③ 返回清晰的 ✅/❌ 文本 |
| LibraryServer.java | ① 设置 java.rmi.server.hostname；② 导出实现并调用 createRegistry；③ rebind 发布服务 |
| LibraryClient.java | ① 查 Registry 获取 Stub；② 命令行菜单循环；③ 所有调用捕获并打印返回字符串 |
| LibrarySystemTest.java | ① CountDownLatch 同时发起 5 线程抢同一本书；② 覆盖借阅/归还/错误场景；③ 最终断言全部“可借” |

**五、实验环境与部署步骤**

# 1. 编译

cd library\_rmi

mkdir -p out

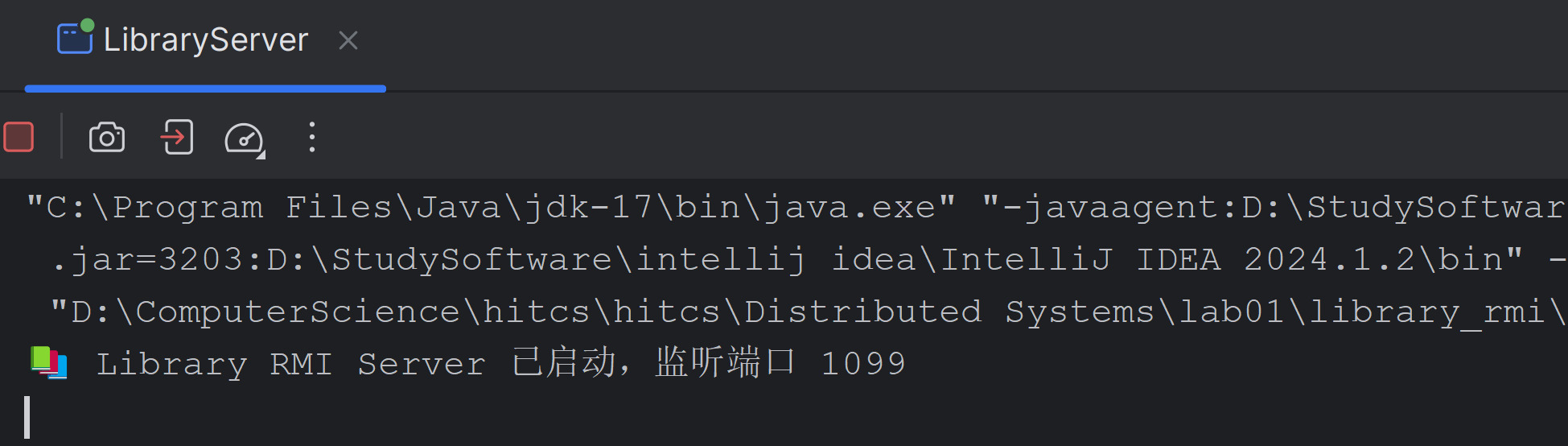
find src -name "\*.java" > sources.txt

javac -encoding UTF-8 -d out @sources.txt

（或者直接在IDEA内运行）

# 2. 启动服务器（终端 1）

java -cp out server.LibraryServer



# 控制台: 📚 Library RMI Server 已启动，监听端口 1099

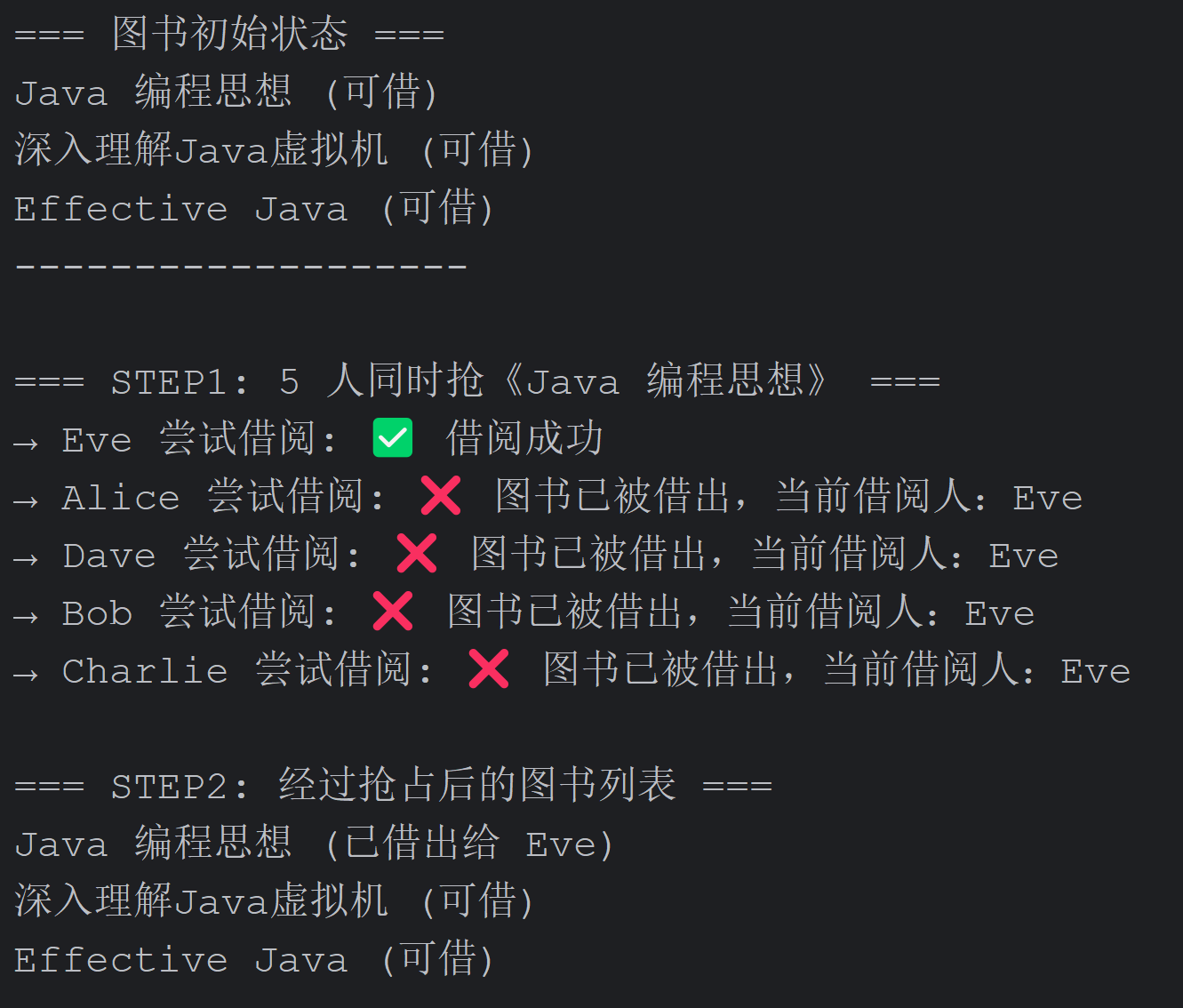
# 3. 运行自动化脚本（终端 2）

java -cp out test.LibrarySystemTest

**六、测试方案与结果**

| 测试用例 | 触发动作 | 预期 | 实际 |
| --- | --- | --- | --- |
| 并发抢书 | 5 线程同时 borrowBook("Java 编程思想") | 1 成功 4 失败 | ✅ |
| 顺序借阅 | 5 人依次借 5 本不同书 | 全部成功 | ✅ |
| 非借阅人归还 | returnBook(..., "Bob") 但实际借阅人是 Alice | 返回❌信息 | ✅ |
| 借不存在书 | borrowBook("不存在的书", "Charlie") | 返回“未找到图书” | ✅ |
| 归还一致性 | 所有人归还后 listBooks() | 所有行以“(可借)”结束 | ✅ |

部分自动化脚本输出（节选）：



文本

AI 生成的内容可能不正确。

**七、结果分析**

1. 远程调用可靠：所有操作均通过 Stub → Skeleton 完成，未出现 RemoteException。
2. 并发排他成功：synchronized 保证“第一完成、其余失败”，无脏写。
3. 错误处理清晰：业务错误（❌）与网络异常分离，客户端体验友好。
4. 脚本自动化验证：降低了人工点按成本，可作为回归测试基线。

**八、总结**

通过本实验：

* 理解了 RMI 的工作机制 —— Registry 命名、Stub / Skeleton 动态代理、序列化与分布式 GC。
* 完成了一个可并发访问的分布式应用，将本地多线程同步思想平滑迁移到跨 JVM 场景。
* 掌握了自动化测试并发逻辑的写法，为后续复杂系统奠定基础。

RMI 的精髓：把“网络调用”降级为“面向对象的普通方法调用”，让开发者把主要精力放在 业务模型 + 并发安全 上，而非底层协议。