HTTP Socket Server

基于Java Socket API实现的HTTP/1.1客户端和服务器系统。

项目简介

本项目是一个完全基于JDK 17 Socket API实现的HTTP协议客户端和服务器，不依赖任何第三方网络框架。系统支持基本的HTTP协议功能，包括：

- HTTP/1.1协议支持
- GET和POST请求方法
- 多种HTTP状态码（200, 301, 302, 304, 400, 401, 404, 405, 500, 503）
- 长连接（Keep-Alive）支持
- 多种MIME类型（HTML, JSON, TXT, PNG）
- 静态资源服务
- 用户注册和登录API
- 并发请求处理（线程池）
- 命令行和GUI客户端界面

项目成员：

- 组长：冯瑞杰
- 组员: 彭世杰，赖均朗， 李秉轩

功能特性

HTTP服务器

- **多线程处理**: 使用线程池处理并发连接
- **静态资源服务**: 从指定目录提供HTML、JSON、TXT、PNG等文件
- **RESTful API**: 提供用户注册和登录接口
- **长连接支持**: 支持HTTP/1.1 Keep-Alive机制
- **错误处理**: 完善的错误处理和状态码返回
- **线程安全**: 使用ConcurrentHashMap确保用户数据的线程安全

HTTP客户端

- **请求发送**: 支持GET和POST请求
- **重定向处理**: 自动处理301、302、304重定向
- **超时控制**: 连接超时和读取超时设置
- **双界面**: 支持命令行（CLI）和图形界面（GUI）两种模式

系统要求

- JDK 17或更高版本
- Maven 3.6+

项目结构

```
src/
├── main/
│   ├── java/
│   │   └── com/
│   │       └── http/
│   │           ├── protocol/          # HTTP协议组件
│   │           │   ├── HttpRequest.java
│   │           │   ├── HttpResponse.java
│   │           │   ├── HttpStatus.java
│   │           │   └── MimeType.java
│   │           ├── server/            # 服务器组件
│   │           │   ├── HttpServer.java
│   │           │   ├── ConnectionHandler.java
│   │           │   ├── RequestRouter.java
│   │           │   ├── RequestHandler.java
│   │           │   ├── StaticResourceHandler.java
│   │           │   ├── RegisterHandler.java
│   │           │   ├── LoginHandler.java
│   │           │   ├── UserRegistry.java
│   │           │   ├── User.java
│   │           │   ├── Session.java
│   │           │   └── ServerMain.java
│   │           ├── client/            # 客户端组件
│   │           │   ├── HttpClient.java
│   │           │   ├── ClientInterface.java
│   │           │   ├── CliClient.java
│   │           │   ├── GuiClient.java
│   │           │   └── ClientMain.java
│   │           └── util/              # 工具类
│   │               └── JsonParser.java
│   └── resources/
│       └── static/                    # 静态资源目录
│           ├── index.html
│           ├── data.json
│           ├── test.txt
│           └── logo.png
└── test/
    └── java/
        └── com/
            └── http/
                ├── EndToEndTest.java  # 端到端测试
                ├── protocol/          # 协议组件测试
                ├── server/            # 服务器组件测试
                └── client/            # 客户端组件测试
```

编译和运行

编译项目

```
mvn clean compile
```

运行测试

```
# 运行所有测试
mvn test

# 运行端到端测试
mvn test -Dtest=EndToEndTest

# 运行特定测试类
mvn test -Dtest=HttpRequestTest
```

服务器使用

启动服务器

```
# 使用默认端口8080
mvn exec:java -D"exec.mainClass"="com.http.server.ServerMain"

# 指定端口
mvn exec:java -D"exec.mainClass"="com.http.server.ServerMain" -D"exec.args"="9090"
```

或者编译后直接运行：

```
mvn package
java -cp target/http-socket-server-1.0-SNAPSHOT.jar com.http.server.ServerMain
java -cp target/http-socket-server-1.0-SNAPSHOT.jar com.http.server.ServerMain
9090
```

服务器配置

- **默认端口**: 8080
- **线程池大小**: 20
- **连接超时**: 30秒
- **静态资源目录**: `src/main/resources/static`

访问服务器

启动服务器后，可以通过以下方式访问：

- 浏览器访问: `http://localhost:8080/index.html`
- API端点:
  - 注册: `POST http://localhost:8080/api/register`
  - 登录: `POST http://localhost:8080/api/login`

客户端使用

启动命令行客户端

```
# CLI模式（默认）
mvn exec:java -Dexec.mainClass="com.http.client.ClientMain"
```

或者：

```
java -cp target/http-socket-server-1.0-SNAPSHOT.jar com.http.client.ClientMain
```

启动GUI客户端

```
# GUI模式
mvn exec:java -Dexec.mainClass="com.http.client.ClientMain" -Dexec.args="gui"
```

或者：

```
java -cp target/http-socket-server-1.0-SNAPSHOT.jar com.http.client.ClientMain gui
```

CLI客户端使用说明

启动CLI客户端后，按照提示输入：

1. **URL**: 完整的URL（如 `http://localhost:8080/index.html`）
2. **Method**: 请求方法（GET或POST）
3. **Headers**: 请求头（格式：`Key: Value`，输入空行结束）
4. **Body**: 请求体（仅POST请求，输入空行结束）

输入 `exit` 或 `quit` 退出客户端。

GUI客户端使用说明

GUI客户端提供图形界面：

1. 在URL输入框输入完整URL
2. 从下拉菜单选择请求方法（GET/POST）
3. 在请求头文本区输入请求头（每行一个，格式：`Key: Value`）
4. 在请求体文本区输入请求体（仅POST请求需要）
5. 点击"Send"按钮发送请求
6. 响应将显示在下方的响应区域

API接口文档

用户注册

端点: `POST /api/register`

请求头:

```
Content-Type: application/json
```

请求体:

```
{
    "username": "testuser",
    "password": "password123"
}
```

验证规则:

- `username`: 3-20个字符
- `password`: 至少6个字符

成功响应 (200 OK):

```
{
    "success": true,
    "message": "Registration successful"
}
```

失败响应 (400 Bad Request):

```
{
    "success": false,
    "message": "Username already exists"
}
```

用户登录

端点: `POST /api/login`

请求头:

```
Content-Type: application/json
```

请求体:

```
{
    "username": "testuser",
    "password": "password123"
}
```

成功响应 (200 OK):

```
{
    "success": true,
    "message": "Login successful",
    "token": "session-token-uuid"
}
```

失败响应 (401 Unauthorized):

```
{
    "success": false,
    "message": "Invalid username or password"
}
```

使用示例

示例1: 使用curl注册用户

```
curl -X POST http://localhost:8080/api/register \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"alice123"}'
```

示例2: 使用curl登录

```
curl -X POST http://localhost:8080/api/login \
  -H "Content-Type: application/json" \
  -d '{"username":"alice","password":"alice123"}'
```

示例3: 使用curl获取静态资源

```
curl http://localhost:8080/index.html
```

示例4: 使用客户端程序

启动服务器后，运行客户端：

```
# 启动CLI客户端
java -cp target/http-socket-server-1.0-SNAPSHOT.jar com.http.client.ClientMain

# 按提示输入
URL: http://localhost:8080/api/register
Method: POST
Headers (empty line to finish):
Content-Type: application/json

Body (empty line to finish):
{"username":"bob","password":"bob123"}
```

测试说明

单元测试

项目包含完整的单元测试，覆盖所有核心组件：

- **协议组件测试**: HttpRequest, HttpResponse, HttpStatus, MimeType
- **服务器组件测试**: UserRegistry, RequestRouter
- **客户端组件测试**: HttpClient
- **工具类测试**: JsonParser

集成测试

- **ServerIntegrationTest**: 测试服务器的完整功能

端到端测试

- **EndToEndTest**: 测试完整的客户端-服务器交互流程
  - 完整用户流程（注册 -> 登录 -> 访问静态资源）
  - 长连接处理
  - 并发客户端
  - 错误场景
  - 不同MIME类型

运行所有测试

```
mvn test
```

查看测试报告

测试报告位于：`target/surefire-reports/`

技术实现细节

HTTP协议解析

- 手动解析HTTP请求和响应报文
- 支持请求行、状态行、头部和正文的完整解析
- 处理Content-Length和Transfer-Encoding

长连接实现

- 检查`Connection: keep-alive`头
- 在同一Socket上处理多个请求
- 30秒空闲超时自动关闭

并发处理

- 使用固定大小线程池（默认20个线程）
- 每个连接由独立线程处理
- ConcurrentHashMap保证用户数据线程安全

错误处理

- 完善的异常捕获和处理
- 适当的HTTP状态码返回
- 错误日志记录

限制和注意事项

1. **简化的JSON解析**: 使用自定义的简单JSON解析器，仅支持基本的键值对格式
2. **内存存储**: 用户数据存储在内存中，服务器重启后数据丢失
3. **密码存储**: 密码以明文存储（实际项目应使用加密）
4. **HTTPS**: 不支持HTTPS/TLS加密
5. **HTTP/2**: 仅支持HTTP/1.1协议
6. **文件上传**: 不支持multipart/form-data文件上传
7. **压缩**: 不支持gzip等内容压缩

故障排除

端口已被占用

```
Error: Address already in use
```

解决方案: 使用不同的端口或关闭占用该端口的程序

```
# Windows
netstat -ano | findstr :8080
taskkill /PID <PID> /F
```

```
# Linux/Mac
lsof -i :8080
kill -9 <PID>
```

连接超时

可能原因:

- 服务器未启动
- 防火墙阻止连接
- 网络问题

解决方案: 检查服务器状态，确认防火墙设置

测试失败

解决方案:

```
# 清理并重新编译
mvn clean compile test
```