








# 数据库字段比较器

一个高效、可扩展的Spring Boot应用程序，用于比较不同数据库中不同表的字段值，并输出差异结果。

## 功能特性

-  **高性能**: 支持批处理和并行处理，优化大数据量比较
-  **可扩展**: 通过简单的配置文件添加新的比较规则
-  **多数据库支持**: 支持MySQL、PostgreSQL、Oracle等主流数据库
-  **详细报告**: 提供完整的差异分析报告
-  **RESTful API**: 提供丰富的API接口
-  **动态数据源**: 运行时动态管理多个数据源
-  **错误处理**: 完善的异常处理和错误恢复机制

## 技术栈

- Spring Boot 3.2.0
- Java 17
- HikariCP (连接池)
- Spring JDBC
- Maven

## 快速开始

### 1. 环境要求

- Java 17或更高版本

- Maven 3.6+
- 需要比较的数据库实例

## 2. 配置数据库

在 `src/main/resources/application.yml` 中配置您的数据源：

```
comparator:
  # 数据源配置
  data-sources:
    - name: mysql-source
      url: jdbc:mysql://localhost:3306/source_db?
        useSSL=false&serverTimezone=UTC
      username: root
      password: password
      driver-class-name: com.mysql.cj.jdbc.Driver

    - name: postgresql-target
      url: jdbc:postgresql://localhost:5432/target_db
      username: postgres
      password: password
      driver-class-name: org.postgresql.Driver

  # 比较规则配置
  rules:
    - name: user-email-comparison
      description: 比较用户表的邮箱字段
      enabled: true
      source-table:
        data-source: mysql-source
        table-name: users
      target-table:
        data-source: postgresql-target
        table-name: users
      key-field: user_id
      compare-field: email
      where-condition: status = 'ACTIVE'
```

### 3. 运行应用

```
# 编译项目
mvn clean compile

# 运行应用
mvn spring-boot:run

# 或者打包后运行
mvn clean package
java -jar target/database-field-comparator-1.0.0.jar
```

应用将在 `http://localhost:8080` 启动。

## 配置说明

### 数据源配置

每个数据源包含以下属性：

| 属性                | 描述       | 必填 |
|-------------------|----------|----|
| name              | 数据源唯一标识  | 是  |
| url               | 数据库连接URL | 是  |
| username          | 数据库用户名   | 是  |
| password          | 数据库密码    | 是  |
| driver-class-name | JDBC驱动类名 | 是  |
| properties        | 额外的连接属性  | 否  |

## 比较规则配置

每个比较规则包含以下属性：

| 属性              | 描述      | 必填       |
|-----------------|---------|----------|
| name            | 规则唯一标识  | 是        |
| description     | 规则描述    | 否        |
| enabled         | 是否启用规则  | 否，默认true |
| source-table    | 源表配置    | 是        |
| target-table    | 目标表配置   | 是        |
| key-field       | 主键字段名   | 是        |
| compare-field   | 要比较的字段名 | 是        |
| where-condition | WHERE条件 | 否        |

## 表配置

每个表配置包含以下属性：

| 属性          | 描述    | 必填 |
|-------------|-------|----|
| data-source | 数据源名称 | 是  |
| table-name  | 表名    | 是  |
| schema      | 模式名   | 否  |

## 性能配置

| 属性               | 描述       | 默认值  |
|------------------|----------|------|
| batch-size       | 批处理大小    | 1000 |
| enable-parallel  | 是否启用并行处理 | true |
| thread-pool-size | 线程池大小    | 10   |

## API接口

### 1. 执行所有比较规则

```
POST /api/comparison/execute-all
```

响应示例:

```
[
  {
    "ruleName": "user-email-comparison",
    "ruleDescription": "比较用户表的邮箱字段",
    "startTime": "2025-06-27 15:55:19",
    "endTime": "2025-06-27 15:55:20",
    "executionTime": 1000,
    "totalRecords": 1000,
    "differenceCount": 5,
    "sourceOnlyCount": 2,
    "targetOnlyCount": 1,
    "valueDifferenceCount": 2,
    "status": "SUCCESS",
    "differences": [
      {
        "keyValue": 123,
        "type": "VALUE_DIFFERENT",
        "sourceValue": "old@email.com",
        "targetValue": "new@email.com",
        "fieldName": "email"
      }
    ]
  }
]
```

## 2. 执行单个比较规则

```
POST /api/comparison/execute/{ruleName}
```

### 3. 批量执行指定规则

```
POST /api/comparison/execute-batch
```

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

```
["user-email-comparison", "product-price-comparison"]
```

### 4. 获取所有规则信息

```
GET /api/comparison/rules
```

### 5. 验证数据源连接

```
GET /api/comparison/validate-connections
```

### 6. 健康检查

```
GET /api/comparison/health
```

## 使用示例

### 添加新的比较规则

1. 在 `application.yml` 中添加新的数据源（如果需要）：



```
comparator:
  data-sources:
    - name: new-database
      url: jdbc:mysql://newhost:3306/newdb
      username: user
      password: pass
      driver-class-name: com.mysql.cj.jdbc.Driver
```

#### 1. 添加新的比较规则:

```
comparator:
  rules:
    - name: product-price-comparison
      description: 比较产品价格
      enabled: true
      source-table:
        data-source: mysql-source
        table-name: products
      target-table:
        data-source: new-database
        table-name: products
      key-field: product_id
      compare-field: price
      where-condition: active = 1
```

#### 1. 重启应用，新规则即可生效。

## 性能优化建议

1. **合理设置批处理大小:** 根据内存和网络情况调整 `batch-size`
2. **使用WHERE条件:** 通过 `where-condition` 减少比较的数据量
3. **数据库索引:** 确保 `key-field` 和 `compare-field` 有适当的索引
4. **并行处理:** 对于多个规则，启用 `enable-parallel` 可提高效率

5. **连接池配置**: 根据并发需求调整连接池参数

## 监控和日志

应用提供了详细的日志输出，可以通过以下方式查看：

```
# 查看实时日志
tail -f logs/database-comparator.log

# 查看错误日志
grep ERROR logs/database-comparator.log
```

## 扩展开发

### 添加新的数据库支持

1. 在 `pom.xml` 中添加相应的JDBC驱动依赖
2. 在 `SqlBuilder.java` 中添加新数据库类型的SQL方言支持
3. 更新配置文档和示例

### 自定义比较逻辑

1. 继承或实现 `FieldComparisonService`
2. 重写 `performComparison` 方法
3. 注册为Spring Bean

### 添加新的输出格式

1. 创建新的结果处理器
2. 在控制器中添加新的端点
3. 支持不同的响应格式（Excel、CSV等）

# 故障排除

## 常见问题

### 1. 数据库连接失败

- 检查数据库URL、用户名、密码是否正确
- 确认数据库服务是否运行
- 检查防火墙和网络连接

### 2. 表或字段不存在

- 验证表名和字段名的大小写
- 检查schema配置是否正确
- 确认数据库用户是否有访问权限

### 3. 内存不足

- 减少 `batch-size` 值
- 增加JVM堆内存: `-Xmx2g`
- 使用WHERE条件限制数据量

### 4. 性能问题

- 为比较字段添加数据库索引
- 启用并行处理
- 优化WHERE条件

## 日志级别配置

```
logging:
  level:
    com.minimax.comparator: DEBUG
    org.springframework.jdbc: DEBUG
```

## 许可证

本项目基于MIT许可证开源。

## 贡献

欢迎提交Issue和Pull Request来帮助改进这个项目。

## 作者

MiniMax Agent - 2025年6月27日