读写分离模块功能需求文档

## 文档信息

| 日期 | 版本 | 作者 | 说明 |
| --- | --- | --- | --- |
| 2024-03-21 | 1.0 | 需求分析组 |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 需求介绍

## 背景说明

随着业务数据量的快速增长和并发访问量的增加，单一数据库实例已无法满足高并发读写需求。读写分离是提高数据库性能和可用性的重要技术手段，通过将读操作和写操作分离到不同的数据库实例来提升整体性能。

## 需求价值

读写分离模块能够有效分散数据库压力，提高系统的并发处理能力和响应速度，同时提供故障转移和负载均衡功能，确保系统的高可用性。

# 功能需求

## 功能列表

2.1.1 读写请求路由：自动识别并路由读写请求到相应的数据库实例

2.1.2 负载均衡：支持多种负载均衡算法，如轮询、权重、最少连接等

2.1.3 故障检测与转移：实时监控数据库状态，自动进行故障转移

2.1.4 连接池管理：管理到主从数据库的连接池

2.1.5 数据一致性保证：处理主从延迟，确保数据读取的一致性

2.1.6 配置管理：支持动态配置主从数据库信息

## 功能详细描述

### 2.2.1 核心功能

1. **功能优先级**：高
2. **实现复杂度**：中等
3. **性能要求**：高性能、低延迟
4. **可靠性要求**：99.9%可用性

### 2.2.2 扩展功能

1. **功能优先级**：中
2. **实现复杂度**：中等
3. **兼容性要求**：向后兼容
4. **可维护性要求**：模块化设计

# 非功能需求

## 功能依赖

### 3.1.1 系统依赖

1. 操作系统：Linux/Windows/macOS
2. 编译器：GCC 或 Clang
3. 标准库：C++17 标准库

### 3.1.2 第三方依赖

1. 日志库：可选集成 spdlog、log4cxx
2. 网络库：可选集成 boost::asio
3. 序列化库：可选集成 protobuf、json

## 资源需求

### 3.2.1 硬件资源

1. **CPU**：最低2核，推荐4核以上
2. **内存**：最低4GB，推荐8GB以上
3. **存储**：最低10GB可用空间
4. **网络**：千兆网络接口

### 3.2.2 软件资源

1. **操作系统**：Linux, Windows, macOS
2. **运行时环境**：C++17 标准库
3. **开发工具**：CMake, Git

## 性能需求

### 3.3.1 响应时间

1. **平均响应时间**：< 10ms
2. **95%响应时间**：< 50ms
3. **99%响应时间**：< 100ms
4. **超时时间**：< 5000ms

### 3.3.2 吞吐量

1. **并发用户数**：支持1000+并发
2. **事务处理量**：> 10000 TPS
3. **数据处理量**：> 100MB/s
4. **查询处理量**：> 50000 QPS

## 可用性需求

### 3.4.1 系统可用性

1. **可用性指标**：99.9%
2. **平均故障时间**：< 4小时/月
3. **恢复时间目标**：< 30分钟
4. **故障检测时间**：< 5分钟

### 3.4.2 容错能力

1. **故障自动恢复**：支持
2. **数据备份**：自动备份
3. **负载均衡**：动态负载均衡
4. **降级处理**：优雅降级

## 安全性需求

### 3.5.1 数据安全

1. **数据加密**：支持AES-256加密
2. **传输安全**：支持TLS 1.3
3. **访问控制**：基于角色的访问控制
4. **审计日志**：完整的操作审计

### 3.5.2 系统安全

1. **身份认证**：多因子认证
2. **权限管理**：细粒度权限控制
3. **安全扫描**：定期安全漏洞扫描
4. **入侵检测**：实时入侵检测

## 可扩展性需求

### 3.6.1 水平扩展

1. **节点扩展**：支持动态添加节点
2. **负载分布**：自动负载重分布
3. **数据分片**：支持数据自动分片
4. **服务发现**：自动服务注册发现

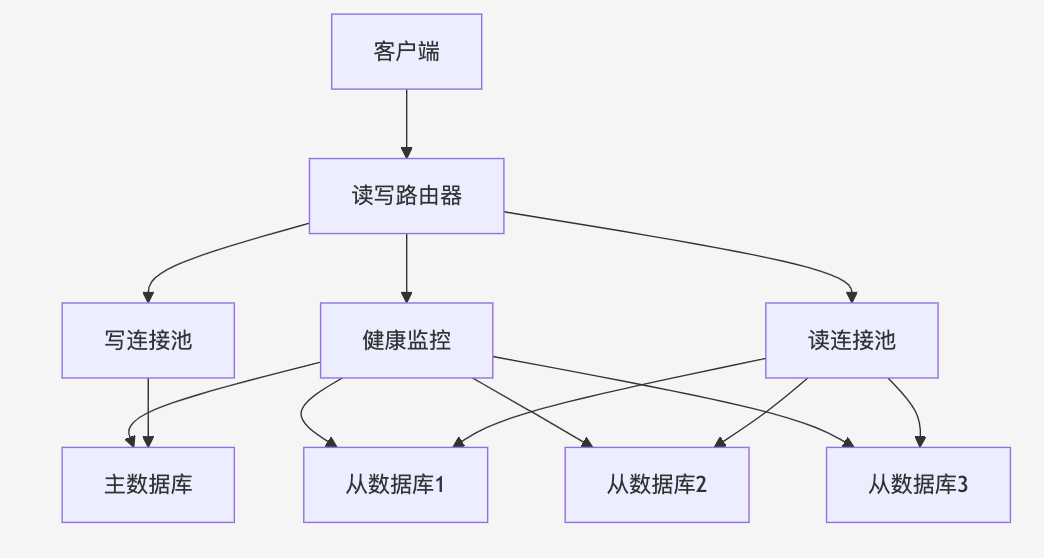
### 3.6.2 垂直扩展

1. **资源扩展**：支持CPU、内存动态扩展
2. **存储扩展**：支持存储容量动态扩展
3. **功能扩展**：插件化架构支持
4. **协议扩展**：支持多种通信协议
5. 验收标准

# 功能设计

## 总体架构

1. 描述rw\_split模块的总体架构图（可插入 Mermaid 或图片）。
2. 列出主要组件及其交互。
3. **示例架构图**：



## 组件划分

| 组件 | 职责 | 关键接口 |
| --- | --- | --- |
| Router | 读写路由器 | route() / selectRead() / selectWrite() |
| ConnectionPool | 连接池 | getConnection() / releaseConnection() |
| HealthMonitor | 健康监控 | checkHealth() / failover() |
| LoadBalancer | 负载均衡器 | balance() / updateWeights() |

## 数据模型

1. 列出关键数据结构及说明。
2. 使用UML类图或Mermaid表示。

## 交互流程

典型业务流程时序图：

