厦門大學

基于分库分表中间件的高并发秒杀系统优化设计与实现

中间件大作业文档

黄勖 22920212204392

张靖源 22920212204492

胡翼翔 22920212204385

2024.06.04

目录

1. 系统介绍

1.1 项目简介

随着电子商务的迅猛发展，促销活动如“秒杀”已成为吸引客户和增加销售额的重要手段。秒杀活动特点是商品数量有限、折扣深，且购买时间限制严格，这常常在极短的时间内引发大量用户的并发访问。这种高并发场景对电商平台的数据处理能力和系统响应速度提出了极高的要求。

传统的单一数据库系统在处理如此高并发的请求时面临着诸多挑战，包括数据访问延迟、事务阻塞、服务器过载等，这些问题极可能导致系统崩溃，从而影响用户体验和企业信誉。为了克服这些技术障碍，必须采用更为高效的数据管理和处理策略。

本项目《基于分库分表中间件的高并发秒杀系统设计与实现》旨在通过实施中间件技术，来实现数据的有效分片和读写分离，提高数据库处理能力，确保系统在高并发条件下的稳定性和高效性。

通过本项目的设计与实现，我们将详细探索分库分表中间件在真实秒杀场景中的应用效果，验证其在分布式数据库环境下对于提升系统性能、增强数据一致性和可靠性的能力。此外，本项目也将为相关领域的开发者和研究人员提供宝贵的实践经验和技术洞见，助力未来电商系统的技术革新和优化。

1.2 项目目标

开发一个高性能的分库分表中间件，专门用于秒杀商城的订单处理。通过合

理的数据库分片策略和高效的数据路由机制，确保在高并发场景下订单系统的稳定运行，实现以下目标：

- **高并发处理能力**：通过中间件的分库分表功能，系统能够在多数据库实例间分散请求压力，提高数据操作的并行度，从而有效应对秒杀场景下的高并发请求。
- **读写分离**：通过配置分库分表中间件的读写分离策略，系统可以将查询操作和事务性写操作分别路由到不同的数据库节点，进一步优化性能。
- **数据一致性保证**：尽管数据分布在多个数据库实例中，分库分表中间件确保跨分片的数据操作维持一致性，对外提供一致的数据视图。
- **容错与可扩展性**：系统设计考虑了容错机制，如数据库实例宕机的情况下自动切换到备用实例。同时，支持动态扩展数据库实例以应对业务增长。
-

1.3 项目背景

分库分表中间件项目旨在解决传统单体数据库在高并发、大数据量场景下的性能瓶颈和可扩展性问题。通过将数据分散到多个数据库实例和表中，该中间件能够显著提高系统的读写性能和扩展能力。本项目主要实现了动态数据源切换和分库分表的功能，通过自定义的哈希路由策略确保数据的均匀分布。

本项目借鉴了 Sharding-jdbc 的分库分表思想，结合了 Spring Boot 的配置和管理机制，提供了灵活的分库分表解决方案。

1.4 项目架构

本项目基于 Spring Boot 框架，采用动态数据源和 MyBatis 插件实现数据

的分库分表功能。核心组件包括动态数据源切换、路由策略实现和 SQL 拦截器。

1.5 核心技术

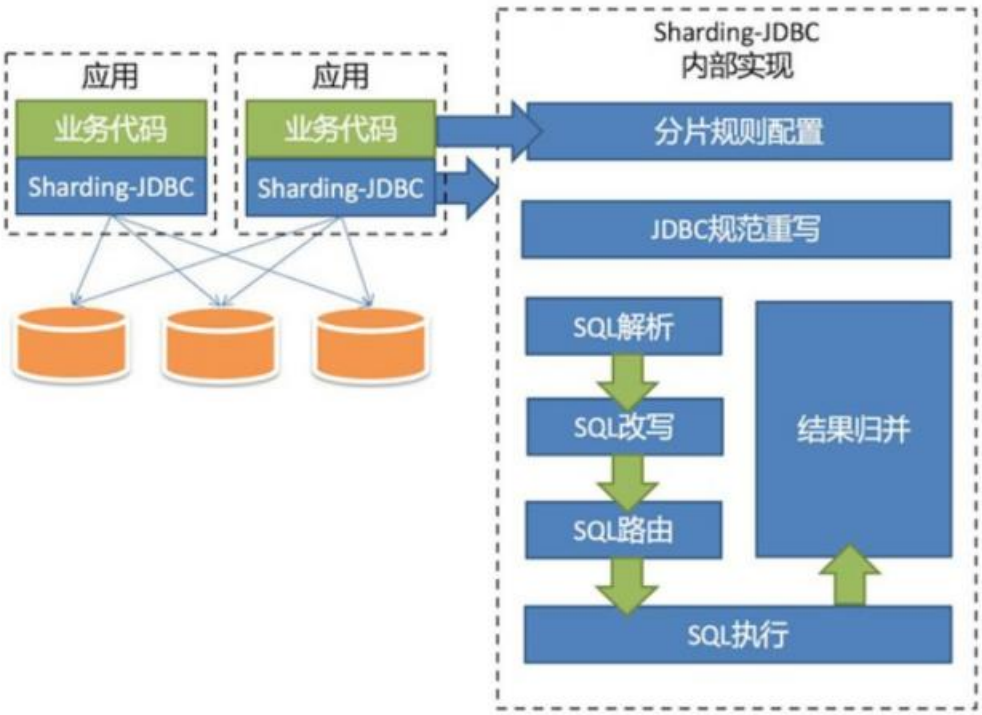
- Spring Boot：提供基础的依赖注入和配置管理。

- 动态数据源切换：通过自定义数据源实现数据源的动态切换。

- 哈希路由策略：确保数据的均匀分布。

- Sharding JDBC：Sharding JDBC 是一款开源的 Java 连接池库，提供了分片和读写分离的功能。在分布式系统中，数据的分片是非常常见的需求，将数据进行分片后可以有效的提高系统性能和可拓展性。Sharding JDBC 以 JDBC 接口为规范，可以被所有基于 JDBC 的应用程序所使用。

Sharding JDBC 的整体架构：



Sharding JDBC 的主要功能包括：

- 分片以及读写分离：支持垂直拆分和水平拆分两种分片方式，同时支持读写分离，可以将读操作和写操作分别指向不同的数据库实例，有效的分担数据库的负载，提高了系统的性能和可靠性。
- 基于 SQL 的路由：Sharding JDBC 支持基于 SQL 语句的路由，通过解析 SQL 语句中的分片键来将 SQL 语句路由到相应的分片数据库实例进行执行。这种路由方式相对于基于数据源的路由更加灵活和高效，可以支持更多的分片场景。
- 事务和连接管理：Sharding JDBC 支持分布式事务，通过在应用程序中使用 XA 接口来实现。同时，它也提供了连接管理功能，可以对连接进行池化和复用，降低了系统开销。
- 简单易用的配置：Sharding JDBC 提供了简单易用的配置方式，用户只需要在配置文件中指定分片规则和读写分离规则即可。
- 负载均衡和故障检测：Sharding JDBC 支持负载均衡和故障检测功能，可以自动检测数据库节点的情况，保证了系统的可用性和容错性。

Sharding-JDBC 的内部实现主要采用代理的方式，通过使用代理对象来替代原始的 JDBC 对象，从而将 SQL 语句路由到相应的分片数据库实例进行执行。Sharding-JDBC 框架将 SQL 解析和路由、连接管理和事务处理等功能进行了封装，大大降低了用户在使用分片数据库时的开发难度和复杂度。

1.6 环境要求

- -**操作系统**: Windows 10 或更高版本、Linux (Ubuntu 18.04 或 CentOS 7 或更高版本)
- **Java**：Java JDK 11 或以上
- **数据库**：MySQL 5.7 或更高版本
- **开发工具**：IntelliJ IDEA 或 Eclipse（带 Spring Boot 插件）
- **其他软件**：Git（用于代码管理和版本控制）

2. 功能介绍

2.1 分库分表功能

DB-Router 数据库路由组件

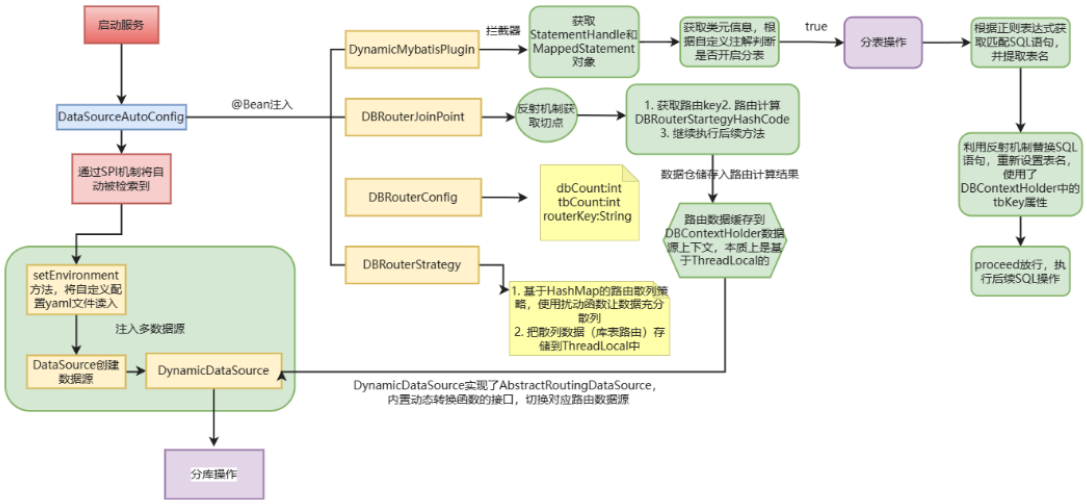
系统架构：基于 AOP、Spring 动态数据源切换、MyBatis 插件开发、散列算法等技术，实现的 SpringBoot Starter 数据库路由组件

核心技术：AOP、AbstractRoutingDataSource、MyBatis Plugin StatementHandler、扰动函数、哈希散列、ThreadLocal

项目描述：此组件项目是为了解决在分库分表场景下，开发一款可以应对自身业务场景多变特性，即支持个性的分库分表、只分库或者只分表以及双字段控制分

库和分表，也可以自定义扩展监控、扫描、策略等规则，同时又能满足简单维护迭代的数据库路由组件。

组件运行流程：



通过自定义的哈希扰动函数实现数据的分库分表，将数据均匀分布到多个数据库和表中，避免单点数据库的性能瓶颈。

2.2 动态数据源切换

利用 Spring 提供的 AbstractRoutingDataSource 类，实现数据源的动态切换，根据当前线程上下文确定使用的数据源。

```
/**
 * 动态数据源获取，获取数据源时，都从这个里面进行获取
 */
3 个用法
public class DynamicDataSource extends AbstractRoutingDataSource {

    0 个用法
    @Override
    protected Object determineCurrentLookupKey() { return "db" + DBContextHolder.getDBKey(); }

}
```

DynamicDataSource.java：继承自 AbstractRoutingDataSource，实现数据源动态切换逻辑。

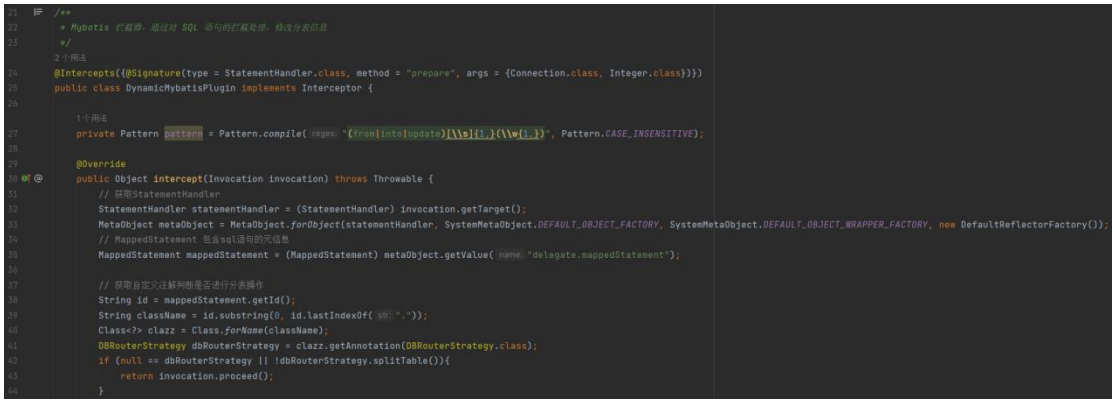
2.3 路由策略

在路由策略的选择上应用了哈希散列原理，使用扰动函数使数据散列地更加均匀。将对应的分库属性的哈希值右移 16 位，高半区和低半区做异或，混合原始哈希吗的高位和低位，来加大低位的随机性。混合后的低位掺杂了高位的部分特征，使高位的信息也被保留下来，同时减少冲突的可能性。

```java
/**
 * 计算方式:
 * size = 库*表的数量
 * idx : 散列到的哪张表
 * dbIdx = idx / dbRouterConfig.getTbCount() + 1;
 * dbIdx : 用于计算哪个库,idx为0-size的值，除以表的数量 = 当前是几号库，又因库是从一号库开始算的，因此需要+1
 * tbIdx : idx - dbRouterConfig.getTbCount() * (dbIdx - 1);用于计算哪个表，
 * idx 可以理解为是第X张表，但是需要落地到是第几个库的第几个表
 * 例子：假设2库8表，idx为14，因此是第二个库的第6个表才是第14张表
 * (dbIdx - 1) 因为库是从1开始算的，因此这里需要-1
 * dbRouterConfig.getTbCount() * (dbIdx - 1) 是为了算出当前库前面的多少张表，也就是要跳过前面的这些表，
 * 然后来计算当前库中的表
 * @param dbKeyAttr 路由字段
 */
1 个用法
@Override
public void doRouter(String dbKeyAttr) {

    // 获取所有表
    int size = dbRouterConfig.getDbCount() * dbRouterConfig.getTbCount();

    // 扰动函数；在 JDK 的 HashMap 中，对于一个元素的存放，需要进行哈希散列。而为了让散列更加均匀，所以添加了扰动函数。
    int idx = (size - 1) & (dbKeyAttr.hashCode() ^ (dbKeyAttr.hashCode() >>> 16));

    // 库表索引；相当于是把一个长条的桶，切割成段，对应分库分表中的库编号和表编号
    // 获取对应的库，库是从1开始算的，因此要在此基础上+1
    int dbIdx = idx / dbRouterConfig.getTbCount() + 1;

    int tbIdx = idx - dbRouterConfig.getTbCount() * (dbIdx - 1);

    // 设置库表信息到上下文,String.format("%02d", dbIdx),数据不为两位的话则在前面补0,这里的策略主要和设置的库表名称有关
    // 例如：库名称为test_01 那就写%02d。表名称user_001 对应%03d
    DBContextHolder.setDBKey(String.format("%02d", dbIdx));
    DBContextHolder.setTBKey(String.format("%03d", tbIdx));
    logger.debug("数据库路由 dbIdx：{} tbIdx：{}",  dbIdx, tbIdx);
}
```

2.4 拦截器实现

通过实现 MyBatis 的拦截器接口，在 SQL 执行前对 SQL 进行修改，实现表名的替换，达到分表的效果。



```java
/**
 * Mybatis 拦截器，通过对 SQL 语句的拦截处理，修改分表信息
 */
2 个用法
@Intercepts({@Signature(type = StatementHandler.class, method = "prepare", args = {Connection.class, Integer.class})})
public class DynamicMybatisPlugin implements Interceptor {

    1 个用法
    private Pattern pattern = Pattern.compile( regex: "(from|into|update)[\\s]{1,}(\\w{1,})", Pattern.CASE_INSENSITIVE);

    @Override
    public Object intercept(Invocation invocation) throws Throwable {
        // 获取StatementHandler
        StatementHandler statementHandler = (StatementHandler) invocation.getTarget();
        MetaObject metaObject = MetaObject.forObject(statementHandler, SystemMetaObject.DEFAULT_OBJECT_FACTORY, SystemMetaObject.DEFAULT_OBJECT_WRAPPER_FACTORY, new DefaultReflectorFactory());
        // MappedStatement 包含sql语句的元信息
        MappedStatement mappedStatement = (MappedStatement) metaObject.getValue( name: "delegate.mappedStatement");

        // 获取自定义注解判断是否进行分表操作
        String id = mappedStatement.getId();
        String className = id.substring(0, id.lastIndexOf( str: "."));
        Class<?> clazz = Class.forName(className);
        DBRouterStrategy dbRouterStrategy = clazz.getAnnotation(DBRouterStrategy.class);
        if (null == dbRouterStrategy || !dbRouterStrategy.splitTable()){
            return invocation.proceed();
        }
```

```java
        // 获取SQL
        BoundSql boundSql = statementHandler.getBoundSql();
        String sql = boundSql.getSql();

        // 替换SQL表名 DB 为 DB_001
        Matcher matcher = pattern.matcher(sql);
        String tableName = null;
        if (matcher.find()) {
            tableName = matcher.group().trim();
        }
        assert null != tableName;
        String replaceSql = matcher.replaceAll( replacement: tableName + "_" + DBContextHolder.getTBKey());

        // 通过反射修改SQL语句
        Field field = boundSql.getClass().getDeclaredField( name: "sql");
        field.setAccessible(true);
        field.set(boundSql, replaceSql);
        field.setAccessible(false);

        return invocation.proceed();
    }

}
```

2.5 数据库动态路由切面

利用 AOP 技术，在目标方法执行前进行数据库路由，通过注解和配置动态获取路由键，根据方法参数值进行路由计算，并在方法执行后清理上下文，确保系统在高并发场景下的数据一致性和性能优化。

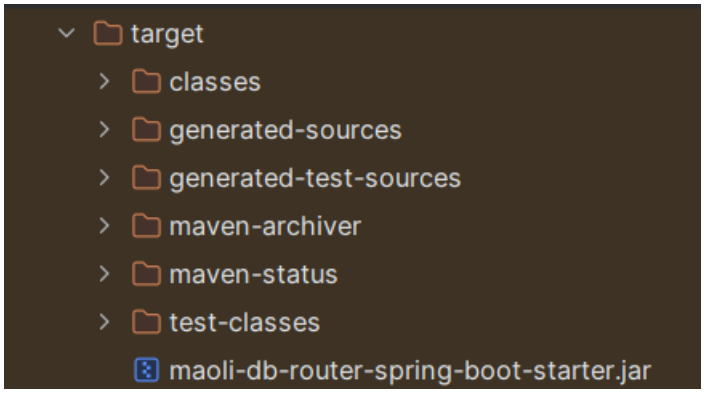
```java
    @Around("aopPoint() && @annotation(dbRouter)")
    public Object doRouter(ProceedingJoinPoint jp, DBRouter dbRouter) throws Throwable {
        String dbKey = dbRouter.key();

        if (StringUtils.isBlank(dbKey) && StringUtils.isBlank(dbRouterConfig.getRouterKey())) {
            throw new RuntimeException("annotation DBRouter key is null! ");
        }
        dbKey = StringUtils.isNotBlank(dbKey) ? dbKey : dbRouterConfig.getRouterKey();
        // 路由属性
        String dbKeyAttr = getAttrValue(dbKey, jp.getArgs());
        // 路由策略
        dbRouterStrategy.doRouter(dbKeyAttr);
        // 返回结果
        try {
            return jp.proceed();
        } finally {
            dbRouterStrategy.clear();
        }
    }
```

```java
    private Method getMethod(JoinPoint jp) throws NoSuchMethodException {
        Signature sig = jp.getSignature();
        MethodSignature methodSignature = (MethodSignature) sig;
        return jp.getTarget().getClass().getMethod(methodSignature.getName(), methodSignature.getParameterTypes());
    }

    1 个用法
    public String getAttrValue(String attr, Object[] args) {
        if (1 == args.length) {
            Object arg = args[0];
            if (arg instanceof String) {
                return arg.toString();
            }
        }

        String filedValue = null;
        for (Object arg : args) {
            try {
                if (StringUtils.isNotBlank(filedValue)) {
                    break;
                }
                filedValue = BeanUtils.getProperty(arg, attr);
            } catch (Exception e) {
                logger.error("获取路由属性值失败 attr: {}", attr, e);
            }
        }
        return filedValue;
    }

}
```

3. 使用说明

3.1 安装步骤

将项目打成 jar 包放入 maven 仓库



在 demo 项目的 pom 中引入

```xml
    <groupId>com.hxstudy.test</groupId>
    <artifactId>demo</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>demo</name>
    <description>自己实现的分库分表中间件示例</description>
    <properties>
        <java.version>1.8</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>com.maoli.dbrouter</groupId>
            <artifactId>db-router-spring-boot-starter</artifactId>
            <version>1.0.0-SNAPSHOT</version>
        </dependency>
```

配置数据源

```yaml
mybatis:
  mapper-locations: classpath:/mapper/*.xml
maoli-db-router:
  jdbc:
    datasource:
      dbCount: 2
      tbCount: 4
      list: db01,db02
      default: db00
      db00:
        driver-class-name: com.mysql.cj.jdbc.Driver
        url: jdbc:mysql://localhost:3306/test?useUnicode=true&characterEncoding=utf8&autoReconnect=true&zeroDateTimeBehavior=convertToNull&serverTimezone=UTC&useSSL=true
        username: root
        password: 123456
      db01:
        driver-class-name: com.mysql.cj.jdbc.Driver
        url: jdbc:mysql://localhost:3306/test_01?useUnicode=true&characterEncoding=utf8&autoReconnect=true&zeroDateTimeBehavior=convertToNull&serverTimezone=UTC&useSSL=true
        username: root
        password: 123456
      db02:
        driver-class-name: com.mysql.cj.jdbc.Driver
        url: jdbc:mysql://localhost:3306/test_02?useUnicode=true&characterEncoding=utf8&autoReconnect=true&zeroDateTimeBehavior=convertToNull&serverTimezone=UTC&useSSL=true
        username: root
        password: 123456
```

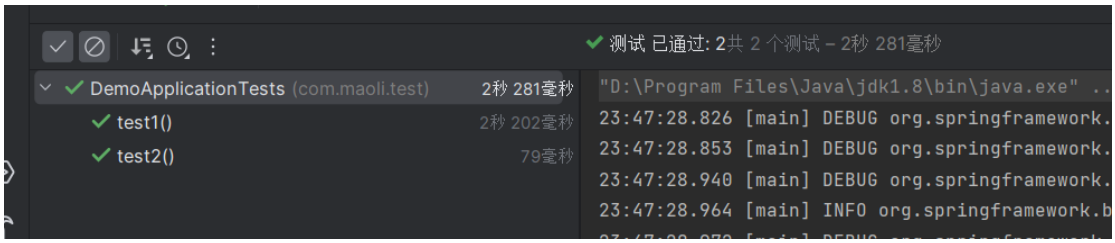
使用注解进行分库分表

```java
5 个用法
@Mapper
@DBRouterStrategy(splitTable = true) // 分表
public interface OrdersSplitTableMapper {

    // 分库
    2 个用法
    @DBRouter(key = "id")
    void insert(Orders order);

    // 分库
    2 个用法
    @DBRouter(key = "id")
    Orders findById(Orders order);
}
```

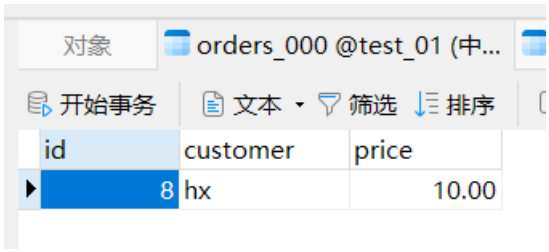
测试用例

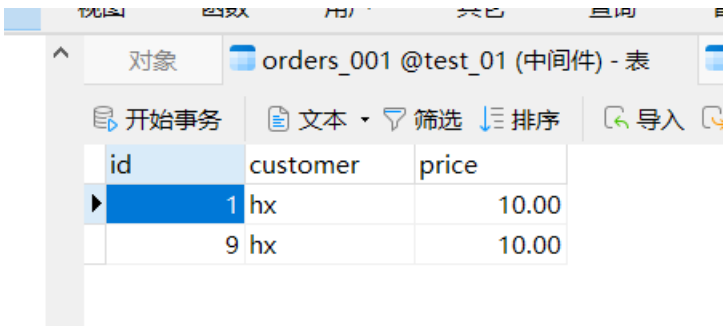
```java
    0 个用法
    @Test
    void test1() {
        for (int i=1;i<=10;i++){
            Orders order=new Orders(i, customer: "hx", price: 10);
            // 分库分表
            ordersSplitTableMapper.insert(order);
        }
    }

    0 个用法
    @Test
    void test2() {
        Orders order=ordersSplitTableMapper.findById(new Orders( id: 7));
        System.out.println(order.getId()==7);
        System.out.println(order.getCustomer().equals("hx"));
        System.out.println(order.getPrice()==10);
    }
```

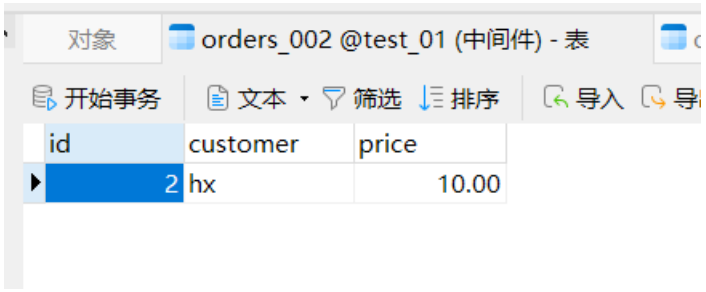
十条数据按照设定规则分表存入两库八表之中

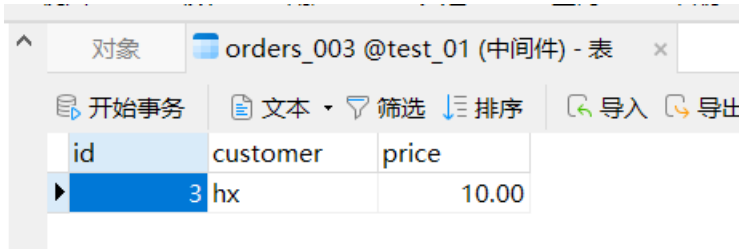

测试 已通过: 2共 2 个测试 – 2秒 281毫秒
DemoApplicationTests (com.maoli.test) 2秒 281毫秒
test1() 2秒 202毫秒
test2() 79毫秒
"D:\Program Files\Java\jdk1.8\bin\java.exe" ..
23:47:28.826 [main] DEBUG org.springframework.
23:47:28.853 [main] DEBUG org.springframework.
23:47:28.940 [main] DEBUG org.springframework.
23:47:28.964 [main] INFO org.springframework.b

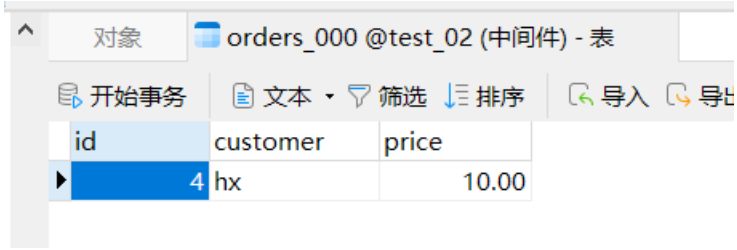
测试结果

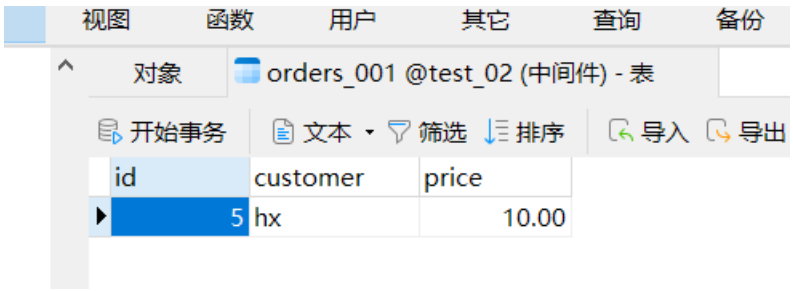

对象
orders_000 @test_01 (中...
开始事务
文本
筛选
排序
id
customer
price
8
hx
10.00

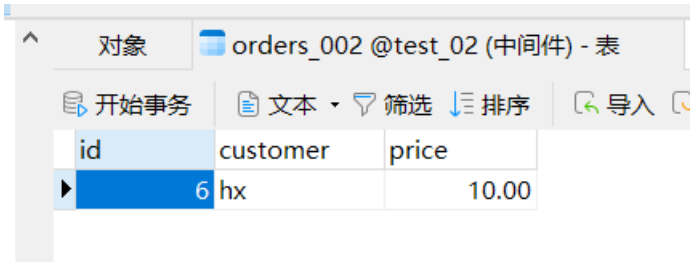

对象
orders_001 @test_01 (中间件) - 表
开始事务
文本
筛选
排序
导入
id
customer
price
1
hx
10.00
9
hx
10.00

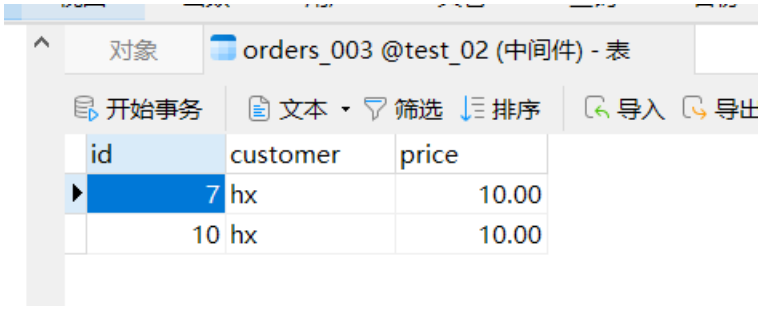

对象
orders_002 @test_01 (中间件) - 表
开始事务
文本
筛选
排序
导入
id
customer
price
2
hx
10.00


对象
orders_003 @test_01 (中间件) - 表
开始事务
文本
筛选
排序
导入
id
customer
price
3
hx
10.00


对象
orders_000 @test_02 (中间件) - 表
开始事务
文本
筛选
排序
导入
id
customer
price
4
hx
10.00







同时分库分表的查询也正常操作

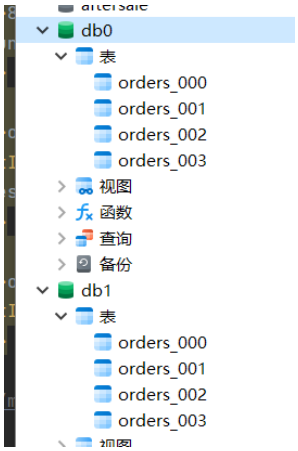
4. 对比测试

4.1 Sharding-jdbc 安装

（1）导入依赖

```xml
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.2.18</version>
        </dependency>

        <!-- sharding-jdbc -->
        <dependency>
            <groupId>org.apache.shardingsphere</groupId>
            <artifactId>sharding-jdbc-spring-boot-starter</artifactId>
            <version>4.0.0-RC1</version>
        </dependency>
    </dependencies>
```

（2）创建数据库 db0 db1、分别创建四张表



（3）引入依赖

```properties
mybatis.mapper-locations=classpath:/mapper/**/*Mapper.xml

spring.shardingsphere.props.sql.show=true
spring.shardingsphere.datasource.names=db0,db1

spring.shardingsphere.datasource.db0.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.db0.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.db0.url= jdbc:mysql://localhost:3306/db0?characterEncoding=utf-8&useSSL=false
spring.shardingsphere.datasource.db0.username=root
spring.shardingsphere.datasource.db0.password=123456

spring.shardingsphere.datasource.db1.type=com.alibaba.druid.pool.DruidDataSource
spring.shardingsphere.datasource.db1.driver-class-name=com.mysql.cj.jdbc.Driver
spring.shardingsphere.datasource.db1.url= jdbc:mysql://localhost:3306/db1?characterEncoding=utf-8&useSSL=false
spring.shardingsphere.datasource.db1.username=root
spring.shardingsphere.datasource.db1.password=123456

spring.shardingsphere.sharding.tables.orders.actual-data-nodes=db$->{0..1}.orders_00$->{0..3}
spring.shardingsphere.sharding.tables.orders.database-strategy.standard.sharding-column=id
spring.shardingsphere.sharding.tables.orders.database-strategy..standard.precise-algorithm-class-name=com.maoli.test.algorithm.MyPreciseDbShardingAlgorithm

spring.shardingsphere.sharding.tables.orders.table-strategy.standard.sharding-column=id
spring.shardingsphere.sharding.tables.orders.table-strategy.standard.precise-algorithm-class-name=com.maoli.test.algorithm.MyPreciseTableShardingAlgorithm

spring.shardingsphere.sharding.default-data-source-name=db0
```

（5）分库算法

```java
package com.maoli.test.algorithm;

import ...

0 个用法
public class MyPreciseDbShardingAlgorithm implements PreciseShardingAlgorithm<Integer> {
    0 个用法
    private final Logger logger = LoggerFactory.getLogger(this.getClass());

    0 个用法
    @Override
    public String doSharding(Collection<String> collection, PreciseShardingValue<Integer> preciseShardingValue) {
        int dbNo = preciseShardingValue.getValue() % 2;

        for (String dbName : collection) {
            if (dbName.endsWith(dbNo + "")) {
                return dbName;
            }
        }

        throw new UnsupportedOperationException();
    }
}
```

（6）分表算法

```java
package com.maoli.test.algorithm;

import ...

0 个用法
public class MyPreciseTableShardingAlgorithm implements PreciseShardingAlgorithm<Integer> {

    0 个用法
    @Override
    public String doSharding(Collection<String> collection, PreciseShardingValue<Integer> preciseShardingValue) {
        int tableNo = preciseShardingValue.getValue() % 4;

        for (String dbName : collection) {
            if (dbName.endsWith(tableNo + "")) {
                return dbName;
            }
        }

        throw new UnsupportedOperationException();
    }
}
```

4.2 测试场景

对比实验设计了四个不同项目实现，并为每种项目实现设计了不同的测试场景进行横向和纵向对比。

三个不同项目实现分别是：简单的 demo 实例，在 demo 上加入我们实现的分库分表中间件实例和使用 Sharding JDBC 中间件的实例。

测试场景：日常场景（较少并发度情况），并发度相较增加情况，高并发情况（100 并发度/s,200 线程/s,500 并发度/s,1000 并发度/s,1500 并发度/s,）

测试功能点：增加订单，根据订单 id 查询订单

POST http://localhost:8080/order

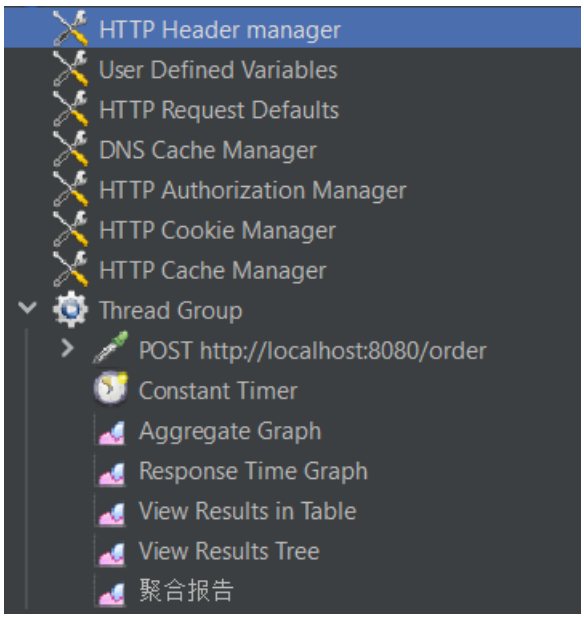
GET http://localhost:8080/order/{id}

4.3 测试

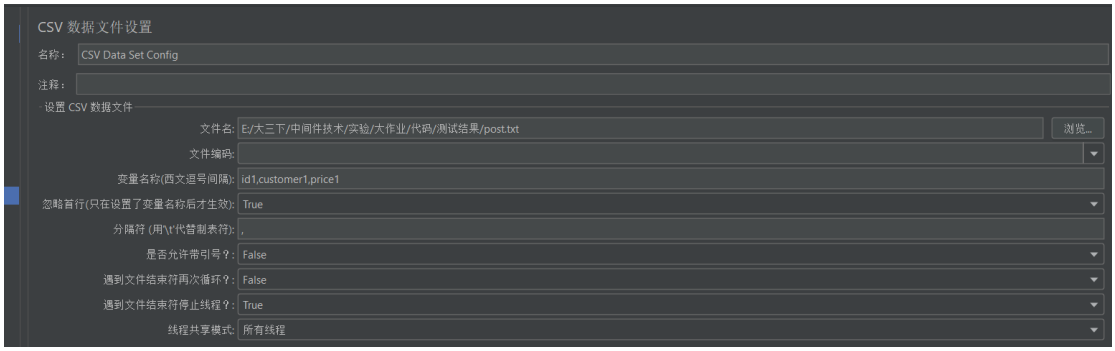
使用测试工具：jemter

测试配置：

POST http://localhost:8080/order 增加订单

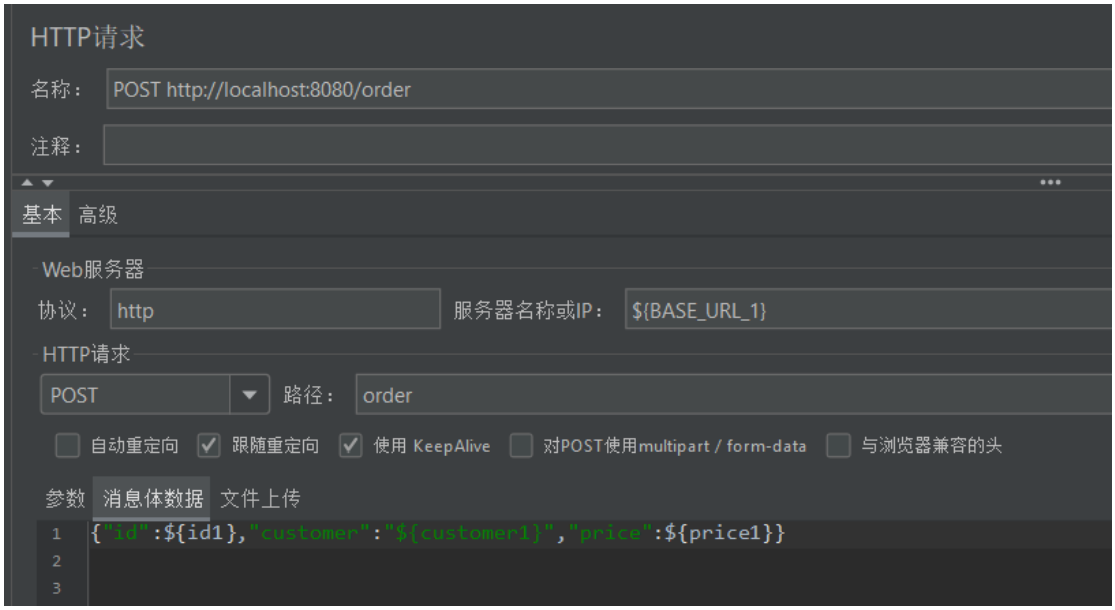


参数化：准备了一万多订单数据

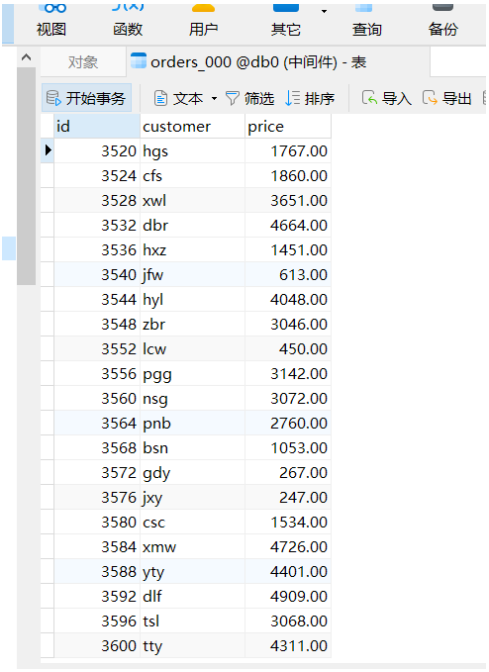


为减少系统预热对实验数据的影响设置定时器

测试的 HTTP Request：

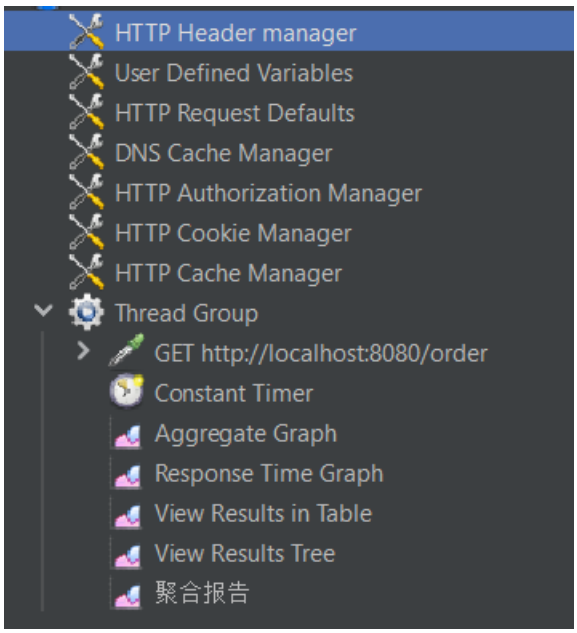


能够正常 POST：



id	customer	price
3520	hgs	1767.00
3524	cfs	1860.00
3528	xwl	3651.00
3532	dbr	4664.00
3536	hxz	1451.00
3540	jfw	613.00
3544	hyl	4048.00
3548	zbr	3046.00
3552	lcw	450.00
3556	pgg	3142.00
3560	nsg	3072.00
3564	pnb	2760.00
3568	bsn	1053.00
3572	gdy	267.00
3576	jxy	247.00
3580	csc	1534.00
3584	xmw	4726.00
3588	yty	4401.00
3592	dlf	4909.00
3596	tsl	3068.00
3600	tty	4311.00

GET http://localhost:8080/order/{id} 根据订单 id 查找 id



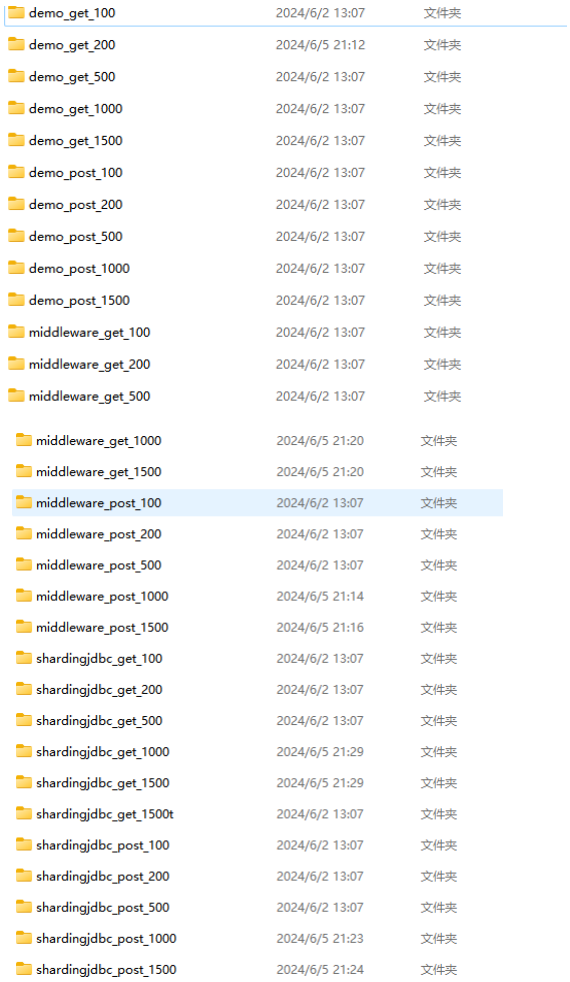
参数化：准备一万多条订单 id 准备查询

测试的 HTTP Request：

正式测试：

设置线程数分别为 100 线程/s,200 线程/s,500 线程/s,1000 线程/s,1500 线程/s 四种不同情况的场景来分别表示不同并发度情况下的场景

生成测试报告：



测试结果分析：

POST http://localhost:8080/order：

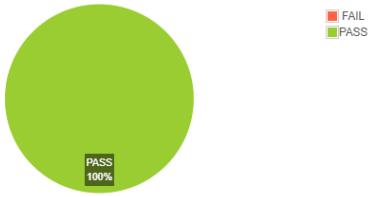
100 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.998	500 ms	1 sec 500 ms	Total
0.998	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

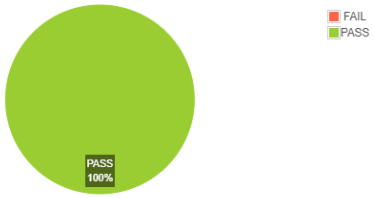
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	12.29	2	605	3.00	5.00	5.00	404.88	100.05	11.82	54.06
POST http://localhost:8080/order	2000	0	0.00%	12.29	2	605	3.00	5.00	5.00	404.88	100.05	11.82	54.06

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

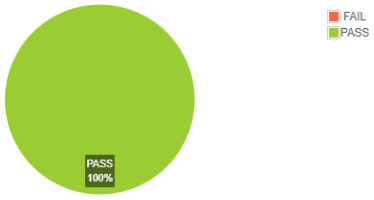
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	3.27	2	10	3.00	4.00	4.00	4.00	99.84	11.80	53.89
POST http://localhost:8080/order	1000	0	0.00%	3.27	2	10	3.00	4.00	4.00	4.00	99.84	11.80	53.89

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.993	**500 ms**	**1 sec 500 ms**	**Total**
0.993	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**1000**	**0**	**0.00%**	**35.64**	**12**	**647**	**15.00**	**19.00**	**163.10**	**559.84**	**99.61**	**11.77**	**53.77**
POST http://localhost:8080/order	1000	0	0.00%	35.64	12	647	15.00	19.00	163.10	559.84	99.61	11.77	53.77

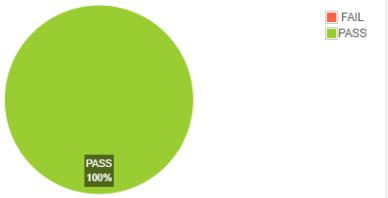
200 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	**500 ms**	**1 sec 500 ms**	**Total**
1.000	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

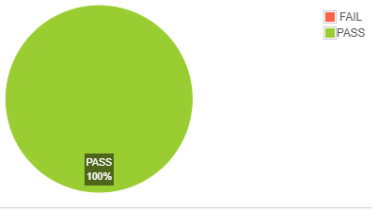
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2000**	**0**	**0.00%**	**3.14**	**2**	**28**	**3.00**	**4.00**	**4.00**	**4.00**	**199.68**	**23.60**	**107.88**
POST http://localhost:8080/order	2000	0	0.00%	3.14	2	28	3.00	4.00	4.00	4.00	199.68	23.60	107.88

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

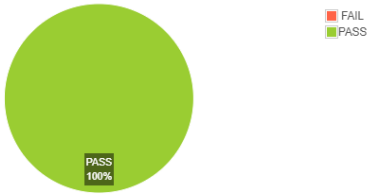
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	14.23	11	40	14.00	15.00	16.00	23.00	198.79	23.49	107.40
POST http://localhost:8080/order	2000	0	0.00%	14.23	11	40	14.00	15.00	16.00	23.00	198.79	23.49	107.40

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	3.43	2	16	3.00	4.00	4.00	5.00	199.48	23.57	107.78
POST http://localhost:8080/order	2000	0	0.00%	3.43	2	16	3.00	4.00	4.00	5.00	199.48	23.57	107.78

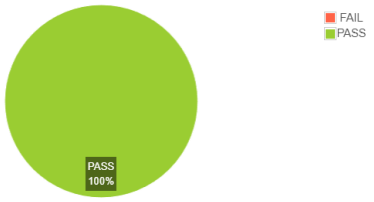
500 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.934	500 ms	1 sec 500 ms	Total
0.934	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

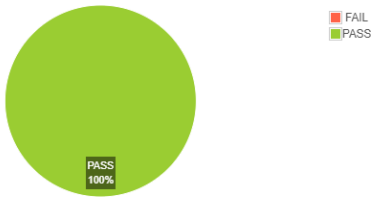
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	5000	0	0.00%	138.94	2	1850	4.00	713.90	964.00	1229.99	499.50	59.02	270.03
POST http://localhost:8080/order	5000	0	0.00%	138.94	2	1850	4.00	713.90	964.00	1229.99	499.50	59.02	270.03

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

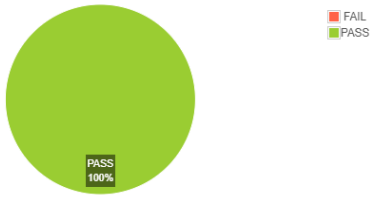
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	5000	0	0.00%	18.52	10	124	17.00	24.00	32.95	56.00	498.75	58.93	269.63
POST http://localhost:8080/order	5000	0	0.00%	18.52	10	124	17.00	24.00	32.95	56.00	498.75	58.93	269.63

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	5000	0	0.00%	3.31	1	45	3.00	4.00	4.00	13.00	499.50	59.02	270.03
POST http://localhost:8080/order	5000	0	0.00%	3.31	1	45	3.00	4.00	4.00	13.00	499.50	59.02	270.03

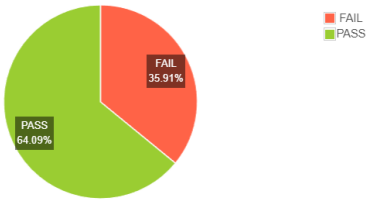
1000 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.608	500 ms	1 sec 500 ms	Total
0.608	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

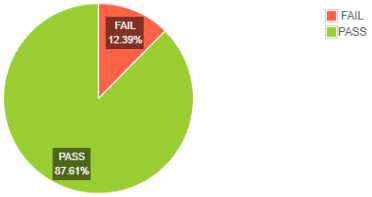
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	10000	3591	35.91%	417.71	3	2798	17.00	1502.90	1788.00	2398.99	936.86	876.69	324.55
POST http://localhost:8080/order	10000	3591	35.91%	417.71	3	2798	17.00	1502.90	1788.00	2398.99	936.86	876.69	324.55

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.156	**500 ms**	**1 sec 500 ms**	**Total**
0.156	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

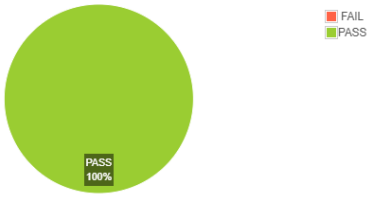
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**1239**	**12.39%**	**2254.98**	**0**	**7658**	**2239.50**	**4231.00**	**4554.00**	**6072.85**	**622.70**	**249.02**	**297.93**
POST http://localhost:8080/order	10000	1239	12.39%	2254.98	0	7658	2239.50	4231.00	4554.00	6072.85	622.70	249.02	297.93

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.937	**500 ms**	**1 sec 500 ms**	**Total**
0.937	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**0**	**0.00%**	**126.59**	**1**	**1422**	**3.00**	**685.00**	**913.00**	**1083.00**	**999.90**	**118.15**	**540.27**
POST http://localhost:8080/order	10000	0	0.00%	126.59	1	1422	3.00	685.00	913.00	1083.00	999.90	118.15	540.27

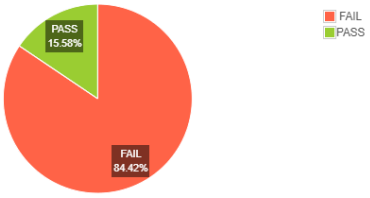
1500 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.156	**500 ms**	**1 sec 500 ms**	**Total**
0.156	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

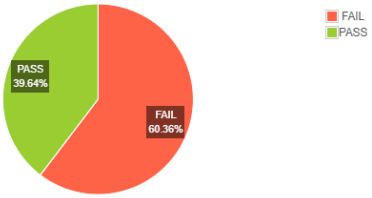
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**15000**	**12663**	**84.42%**	**128.66**	**2**	**515**	**97.00**	**302.00**	**385.00**	**470.00**	**999.13**	**1934.32**	**84.10**
POST http://localhost:8080/order	15000	12663	84.42%	128.66	2	515	97.00	302.00	385.00	470.00	999.13	1934.32	84.10

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.313	**500 ms**	**1 sec 500 ms**	**Total**
0.313	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

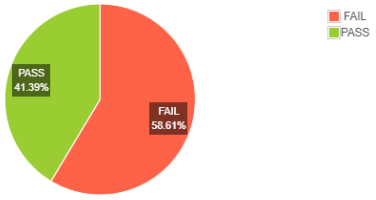
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**15000**	**9054**	**60.36%**	**262.51**	**0**	**4429**	**108.00**	**601.00**	**882.00**	**2451.99**	**913.74**	**1306.70**	**195.82**
POST http://localhost:8080/order	15000	9054	60.36%	262.51	0	4429	108.00	601.00	882.00	2451.99	913.74	1306.70	195.82

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.414	500 ms	1 sec 500 ms	Total
0.414	500 ms	1 sec 500 ms	POST http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	15000	8792	58.61%	69.56	1	734	7.00	276.00	358.00	550.99	999.20	1379.19	223.54
POST http://localhost:8080/order	15000	8792	58.61%	69.56	1	734	7.00	276.00	358.00	550.99	999.20	1379.19	223.54

结论：在并发量较小时，大部分情况下都能完美通过，在并发量为 500 线程/s 的时候，没有使用中间件的 demo 出现 40%的错误，我们认为是运行环境波动的偶然现象。而在并发量较大时，我们实现的中间件性与 Sharding-JDBC 中间件非常相似，相差不到 2%。

GET http://localhost:8080/order：

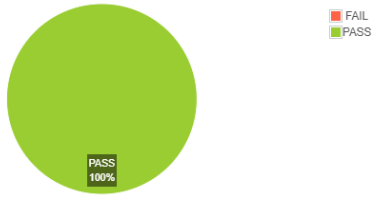
100 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

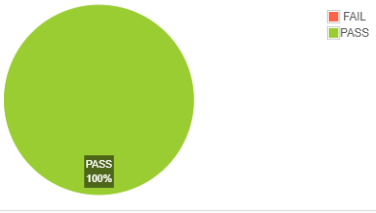
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	15.84	1	513	3.00	4.00	21.50	420.92	99.67	19.91	50.12
GET http://localhost:8080/order	1000	0	0.00%	15.84	1	513	3.00	4.00	21.50	420.92	99.67	19.91	50.12

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.993	500 ms	1 sec 500 ms	Total
0.993	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

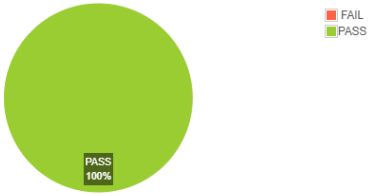
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	35.23	11	654	14.00	18.00	158.75	559.92	99.86	18.66	50.22
GET http://localhost:8080/order	1000	0	0.00%	35.23	11	654	14.00	18.00	158.75	559.92	99.86	18.66	50.22

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	1000	0	0.00%	2.71	1	66	2.00	3.00	3.00	9.96	100.03	13.17	50.30
GET http://localhost:8080/order	1000	0	0.00%	2.71	1	66	2.00	3.00	3.00	9.96	100.03	13.17	50.30

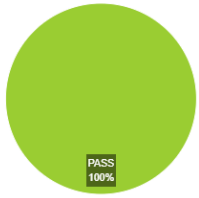
200 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

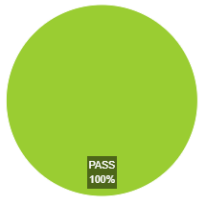
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	1.73	1	4	2.00	2.00	2.00	2.00	199.82	41.61	100.59
GET http://localhost:8080/order	2000	0	0.00%	1.73	1	4	2.00	2.00	2.00	2.00	199.82	41.61	100.59

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

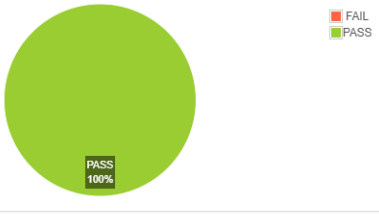
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	2000	0	0.00%	13.60	11	27	13.00	14.00	15.00	23.00	198.93	31.47	100.14
GET http://localhost:8080/order	2000	0	0.00%	13.60	11	27	13.00	14.00	15.00	23.00	198.93	31.47	100.14

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	**500 ms**	**1 sec 500 ms**	**Total**
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**2000**	**0**	**0.00%**	**2.18**	**1**	**10**	**2.00**	**3.00**	**3.00**	**3.00**	**199.48**	**24.93**	**100.42**
GET http://localhost:8080/order	2000	0	0.00%	2.18	1	10	2.00	3.00	3.00	3.00	199.48	24.93	100.42

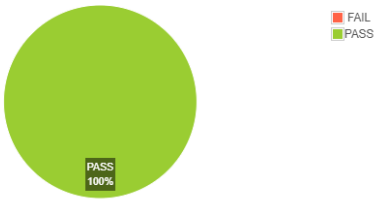
500 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	**500 ms**	**1 sec 500 ms**	**Total**
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

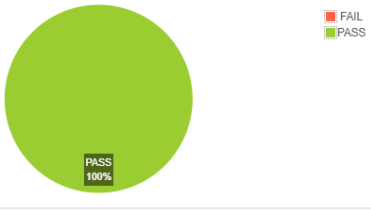
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**5000**	**0**	**0.00%**	**2.03**	**1**	**52**	**2.00**	**3.00**	**3.00**	**4.00**	**500.00**	**74.48**	**251.85**
GET http://localhost:8080/order	5000	0	0.00%	2.03	1	52	2.00	3.00	3.00	4.00	500.00	74.48	251.85

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

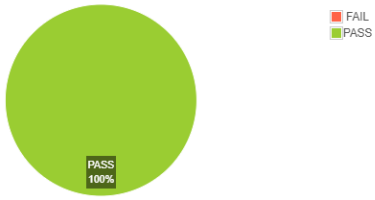
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	5000	0	0.00%	15.83	9	97	15.00	21.00	27.00	43.00	499.25	70.09	251.47
GET http://localhost:8080/order	5000	0	0.00%	15.83	9	97	15.00	21.00	27.00	43.00	499.25	70.09	251.47

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	5000	0	0.00%	2.11	1	43	2.00	3.00	3.00	6.00	500.05	60.60	251.87
GET http://localhost:8080/order	5000	0	0.00%	2.11	1	43	2.00	3.00	3.00	6.00	500.05	60.60	251.87

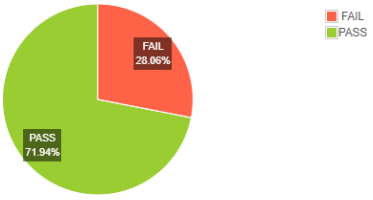
1000 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.718	**500 ms**	**1 sec 500 ms**	**Total**
0.718	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

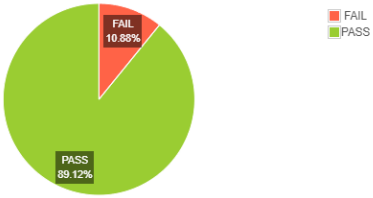
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**2806**	**28.06%**	**94.07**	**1**	**957**	**2.00**	**389.00**	**548.95**	**707.99**	**998.30**	**740.18**	**361.79**
GET http://localhost:8080/order	10000	2806	28.06%	94.07	1	957	2.00	389.00	548.95	707.99	998.30	740.18	361.79

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.281	**500 ms**	**1 sec 500 ms**	**Total**
0.281	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

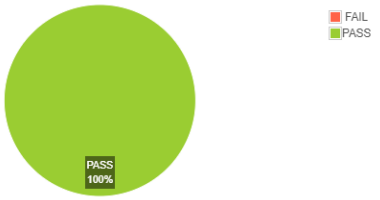
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**10000**	**1088**	**10.88%**	**1587.18**	**0**	**6928**	**1484.50**	**3211.00**	**3566.90**	**4963.98**	**657.16**	**287.16**	**296.32**
GET http://localhost:8080/order	10000	1088	10.88%	1587.18	0	6928	1484.50	3211.00	3566.90	4963.98	657.16	287.16	296.32

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
1.000	500 ms	1 sec 500 ms	Total
1.000	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	10000	0	0.00%	2.00	0	46	2.00	2.00	3.00	9.99	999.00	176.56	503.30
GET http://localhost:8080/order	10000	0	0.00%	2.00	0	46	2.00	2.00	3.00	9.99	999.00	176.56	503.30

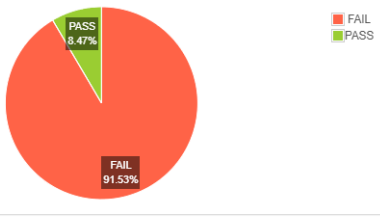
1500 线程/s：

demo：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.085	500 ms	1 sec 500 ms	Total
0.085	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

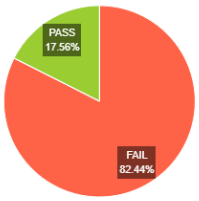
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	15000	13729	91.53%	150.74	0	677	103.00	374.00	465.95	589.00	999.00	2090.58	42.61
GET http://localhost:8080/order	15000	13729	91.53%	150.74	0	677	103.00	374.00	465.95	589.00	999.00	2090.58	42.61

分库分表中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.123	500 ms	1 sec 500 ms	Total
0.123	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

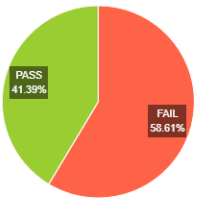
Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	15000	12366	82.44%	433.65	0	5543	352.00	707.00	882.95	3329.99	983.35	1999.17	86.97
GET http://localhost:8080/order	15000	12366	82.44%	433.65	0	5543	352.00	707.00	882.95	3329.99	983.35	1999.17	86.97

Sharding-JDBC 中间件：

APDEX (Application Performance Index)

Apdex	T (Toleration threshold)	F (Frustration threshold)	Label
0.414	500 ms	1 sec 500 ms	Total
0.414	500 ms	1 sec 500 ms	GET http://localhost:8080/order

Requests Summary



Statistics

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	15000	8791	58.61%	66.89	0	636	4.00	240.00	384.00	575.99	999.67	1396.05	208.45
GET http://localhost:8080/order	15000	8791	58.61%	66.89	0	636	4.00	240.00	384.00	575.99	999.67	1396.05	208.45

5 总结与展望

结论：并发量较小时，三种项目的通过率均为 100%，当并发量达到 1000 线程/s 时，与 POST 相比，Sharding JDBC 中间件仍然展示出优异和稳定的性能，而我们的中间件开始出错，但是优于没有使用中间件的 demo。执行 GET 指令时我们的中间件性能明显弱于 POST 指令。

展望:

1.可以发现自己实现的中间件还存在效率上的问题，这可能是我们在aop 的实现上还存在一部分的冗余操作，可以在后续学习shardingjdbc 的思路继续实现。

2.还可以尝试不同的分库分表策略，设置不同的算法来进行分库分表。

在这几次大实验中，我们小组通过编写相关代码，成功地利用中间件相关技术实现了项目的要求，体会到了中间件技术的方便和高效，我们都收获颇丰。