哈尔滨工业大学

**计算学部**

**2022年春季学期**

**《软件架构与中间件》课程**

**实验报告**

**Lab 5&6：分布式系统整合和质量测试实验**

|  |  |  |
| --- | --- | --- |
| **姓名** | **学号** | **联系方式** |
| XXX | NNNNNNN | Email/手机号码 |
| XXX | NNNNNNN | Email/手机号码 |

目 录

[1 实验概述 1](#_Toc104571738)

[1.1 实验目的 1](#_Toc104571739)

[1.2 实验要求 1](#_Toc104571740)

[2 实验内容与过程 1](#_Toc104571741)

[3 结对开发过程记录 2](#_Toc104571742)

[4 实验总结 3](#_Toc104571743)

[5 教师评语 3](#_Toc104571744)

[文档全部完成之后，请在上述区域点击右键，选择“更新域”，在打开的对话框中选择“更新整个目录”]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **学号：** |  | **姓名：** |  | |
| **学号：** |  | **姓名** |  | |
| 实验概述 | | | |  |
| 实验目的  1. 尝试综合运用计算层、数据层和表示层的架构技术 2. 实现满足高性能、高可用、高可靠等质量属性的复杂软件系统 3. 学会分析和构建高效的分布式软件系统 4. 学习针对分布式系统进行质量测试的方法、工具 5. 学习使用诸如Jmeter的系统质量测试工具，掌握其安装与配置的方法 6. 能够根据性能分析与评估报告对系统的架构细节进行调优或整体升级  实验要求  1. 2人结对成组 2. 基于实验2-4在不同层面（计算、数据、表示）上的软件架构技术，对遗留进销存系统进行分布式改造。 3. 按照用户需求，商业业务逻辑，将各部分的内容进行逐步整合，形成一个功能完整，且具有较高可用性、可靠性等的分布式软件系统。 4. 分析改造前后系统的非功能指标。 5. 按照课程中7步骤测试法，对实验5的系统进行质量测试。 6. 分析改造前后系统的质量指标(包括常规性能指标) 7. 关注压力测试 8. 撰写系统集成报告和测试报告 | | | | |
| 实验内容与过程 | | | | |
| * 1. 对遗留系统从三个层面进行全面的分布式改造的设计过程和实现细节。  总体设计 绘图1  因为自己的笔记本性能有限，这里只开了三个虚拟机作为集群，模拟分布式环境。选用的技术有：Nginx（计算层），Sharding-JDBC和Redis（数据层），React（表示层）。  集群上部署的应用架构如上图。  首先一台主机运行Nginx，负责分发静态资源（前端使用React开发的页面），并对请求进行反向代理。然后三台主机都运行一个Tomcat后端（使用SpringBoot开发）。Ningx根据规则（比如ip\_hash）将请求分别分发到这三个后端。在数据层，使用Redis作为数据的缓存，这里部署了单机的Redis在一台主机上，实际上Redis是可以搭集群的，但是这里因为懒精力有限，只使用了单机。然后使用Sharding-JDBC进行分库分表，将数据分别储存在三台主机的MySQL上。  下面介绍整个开发过程： Sharding-JDBC 引入依赖   1. <!--sharding jdbc --> 2. <!--<!– https://mvnrepository.com/artifact/org.apache.shardingsphere/sharding-jdbc-spring-boot-starter –>--> 3. **<dependency>** 4. **<groupId>**org.apache.shardingsphere**</groupId>** 5. **<artifactId>**sharding-jdbc-spring-boot-starter**</artifactId>** 6. **<version>**4.1.1**</version>** 7. **</dependency>** 8. **<dependency>** 9. **<groupId>**org.apache.shardingsphere**</groupId>** 10. **<artifactId>**sharding-jdbc-spring-namespace**</artifactId>** 11. **<version>**4.1.1**</version>** 12. **</dependency>** 13. <!--集成druid连接池--> 14. **<dependency>** 15. **<groupId>**com.alibaba**</groupId>** 16. **<artifactId>**druid**</artifactId>** 17. **<version>**1.2.16**</version>** 18. **</dependency>**   在application.yml中添加以下配置   1. spring: 2. shardingsphere: 3. datasource: 4. names: db1, db2, db3 5. db1: 6. type: com.alibaba.druid.pool.DruidDataSource 7. driver-class-name: com.mysql.cj.jdbc.Driver 8. url: jdbc:mysql://localhost:3306/test1 9. username: test 10. password: test 11. db2: 12. type: com.alibaba.druid.pool.DruidDataSource 13. driver-class-name: com.mysql.cj.jdbc.Driver 14. url: jdbc:mysql://localhost:3306/test2 15. username: test 16. password: test 17. db3: 18. type: com.alibaba.druid.pool.DruidDataSource 19. driver-class-name: com.mysql.cj.jdbc.Driver 20. url: jdbc:mysql://localhost:3306/test3 21. username: test 22. password: test 23. sharding: 24. tables: 25. t\_customer: 26. actual-data-nodes: db$-**>**{1..3}.t\_customer 27. database-strategy: 28. inline: 29. sharding-column: id 30. algorithm-expression: db$-**>**{id % 3 + 1} 31. key-generator: 32. column: id 33. type: SNOWFLAKE 34. t\_order: 35. actual-data-nodes: db$-**>**{1..3}.t\_order 36. database-strategy: 37. inline: 38. sharding-column: id 39. algorithm-expression: db$-**>**{id % 3 + 1} 40. key-generator: 41. column: id 42. type: SNOWFLAKE 43. t\_product: 44. actual-data-nodes: db$-**>**{1..3}.t\_product 45. database-strategy: 46. inline: 47. sharding-column: id 48. algorithm-expression: db$-**>**{id % 3 + 1} 49. key-generator: 50. column: id 51. type: SNOWFLAKE   这里对数据进行水平切片，经数据分散到不同的表中。 接下来按照我个人的理解对这些配置进行讲解。 IMG_256 这里定义数据源，type为数据源的类型，这里集成了druid连接池，所以选择com.alibaba.druid.pool.DruidDataSource。  接下来是分片规则：  IMG_256  actual-data-nodes表示这个table在哪些数据源中有。 algorithm-expression表示按照id对3求余再加1，比如对于id为5的，对应db3，对于id为3的对应db1。 key-generator中的SNOWFLAKE表面按照雪花算法生成id。  这里在开发过程中发现一个问题：  在测试时发现insert的数据，不会将id回显到实体类中，实体类中的id为0，查看mybatis generator生成的xml代码，发现对应语句，如下： IMG_256 按照这样的写法，应该是可以回显到实体类中的  但是经过手动在mysql中执行SELECT LAST\_INSERT\_ID();发现其返回也一直是0，即使对于设置了自增主键的表也是这样。 个人分析，是因为，Sharding-JDBC的key-generator是在程序中生成一个id，再将数据整个插入到数据库中，这个id并不是通过MySQL自己生成的，所以通过这个函数获取到的一直是0。  修改方法如下： 将SELECT LAST\_INSERT\_ID();修改为SELECT MAX( id ) FROM t\_customer即可   1. **<insert** id="insert" parameterType="com.hit.lab.back.bean.Customer"**>** 2. **<selectKey** keyProperty="id" order="AFTER" resultType="java.lang.Long"**>** 3. SELECT MAX( id ) FROM `t\_customer` 4. **</selectKey>** 5. insert into t\_customer (type, name) 6. values (#{type,jdbcType=VARCHAR}, #{name,jdbcType=VARCHAR}) 7. **</insert>**  **Redis** 引入依赖   1. <!-- 集成redis依赖  --> 2. **<dependency>** 3. **<groupId>**org.springframework.boot**</groupId>** 4. **<artifactId>**spring-boot-starter-data-redis**</artifactId>** 5. **</dependency>** 6. <!--连接池--> 7. **<dependency>** 8. **<groupId>**org.apache.commons**</groupId>** 9. **<artifactId>**commons-pool2**</artifactId>** 10. **</dependency>** 11. <!-- https://mvnrepository.com/artifact/com.alibaba.fastjson2/fastjson2 --> 12. **<dependency>** 13. **<groupId>**com.alibaba.fastjson2**</groupId>** 14. **<artifactId>**fastjson2**</artifactId>** 15. **<version>**2.0.32**</version>** 16. **</dependency>** 17. <!-- https://mvnrepository.com/artifact/com.alibaba.fastjson2/fastjson2-extension-spring5 --> 18. **<dependency>** 19. **<groupId>**com.alibaba.fastjson2**</groupId>** 20. **<artifactId>**fastjson2-extension-spring5**</artifactId>** 21. **<version>**2.0.32**</version>** 22. **</dependency>**   这里不仅引入了redis的依赖，还有连接池的依赖，以及fastjson2的依赖。连接池和fastjson2都是为了优化系统性能。连接池的作用不必多说，fastjson2作为Redis序列化的工具可以更快的进行序列化。  配置文件  在application.yml中添加以下配置   1. spring: 2. redis: 3. host: localhost 4. port: 6379 5. #    password: 123456 6. database: 0 7. #    cluster: 8. #      nodes: 10.255.144.115:7001,10.255.144.115:7002,10.255.144.115:7003,10.255.144.115:7004,10.255.144.115:7005,10.255.144.115:7006 9. #      max-redirects: 3 10. lettuce: 11. pool: 12. max-idle: 16 13. max-active: 32 14. min-idle: 8   添加自定义配置   1. **import** com.alibaba.fastjson2.support.spring.data.redis.FastJsonRedisSerializer; 2. **import** org.springframework.beans.factory.annotation.Value; 3. **import** org.springframework.context.annotation.Bean; 4. **import** org.springframework.context.annotation.Configuration; 5. **import** org.springframework.data.redis.cache.RedisCacheConfiguration; 6. **import** org.springframework.data.redis.cache.RedisCacheManager; 7. **import** org.springframework.data.redis.connection.RedisConnectionFactory; 8. **import** org.springframework.data.redis.connection.RedisStandaloneConfiguration; 9. **import** org.springframework.data.redis.connection.lettuce.LettuceConnectionFactory; 10. **import** org.springframework.data.redis.core.RedisTemplate; 11. **import** org.springframework.data.redis.serializer.RedisSerializationContext; 12. **import** org.springframework.data.redis.serializer.StringRedisSerializer; 14. **import** java.time.Duration; 16. /\*\* 17. \* Description: 设置redis配置 18. \*/ 19. @Configuration 20. **public** **class** RedisConfig { 22. @Value("${spring.redis.host}") 23. **private** String redisHost; 25. @Value("${spring.redis.port}") 26. **private** **int** redisPort; 28. @Bean 29. **public** LettuceConnectionFactory redisConnectionFactory() { 30. RedisStandaloneConfiguration configuration = **new** RedisStandaloneConfiguration(redisHost, redisPort); 31. **return** **new** LettuceConnectionFactory(configuration); 32. } 34. @Bean 35. **public** RedisCacheManager cacheManager(RedisConnectionFactory connectionFactory) { 36. FastJsonRedisSerializer<Object> fastJsonRedisSerializer = **new** FastJsonRedisSerializer<>(Object.**class**); 37. StringRedisSerializer stringRedisSerializer = **new** StringRedisSerializer(); 38. RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig() 39. // 默认缓存时间 40. .entryTtl(Duration.ofSeconds(600)) 41. // 设置key的序列化方式 42. .serializeKeysWith(RedisSerializationContext.SerializationPair.fromSerializer(stringRedisSerializer)) 43. // 设置value的序列化方式 44. .serializeValuesWith(RedisSerializationContext.SerializationPair.fromSerializer(fastJsonRedisSerializer)); 45. **return** RedisCacheManager.builder(connectionFactory) 46. .cacheDefaults(config) 47. .transactionAware() 48. .build(); 49. } 51. @Bean 52. **public** RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory redisConnectionFactory) { 53. RedisTemplate<Object, Object> redisTemplate = **new** RedisTemplate<>(); 54. // 使用fastJson序列化 55. // value值的序列化采用fastJsonRedisSerializer 56. redisTemplate.setValueSerializer(**new** FastJsonRedisSerializer<>(Object.**class**)); 57. redisTemplate.setHashValueSerializer(**new** FastJsonRedisSerializer<>(Object.**class**)); 58. // key的序列化采用StringRedisSerializer 59. redisTemplate.setKeySerializer(**new** StringRedisSerializer()); 60. redisTemplate.setHashKeySerializer(**new** StringRedisSerializer()); 61. redisTemplate.setConnectionFactory(redisConnectionFactory); 62. **return** redisTemplate; 63. } 65. }   在redisTemplate这个方法中，指定序列化使用import com.alibaba.fastjson2.support.spring.data.redis.FastJsonRedisSerializer;这个序列化器。  关于Redis在Springboot的使用：  封装工具类：   1. /\*\* 2. \* Redis 工具类 3. \*/ 4. @Service 5. **public** **class** RedisUtils { 6. @Autowired 7. **private** RedisTemplate redisTemplate; 9. **private** **static** **double** size = Math.pow(2, 32); 11. /\*\* 12. \* 写入缓存 13. \* @param key 14. \* @param offset 15. \* @param isShow 16. \* @return result 17. \*/ 18. **public** **boolean** setBit(String key, **long** offset, **boolean** isShow) { 19. **boolean** result = **false**; 20. **try** { 21. ValueOperations<Serializable, Object> operations = redisTemplate.opsForValue(); 22. operations.setBit(key, offset, isShow); 23. result = **true**; 24. } **catch** (Exception e) { 25. e.printStackTrace(); 26. } 27. **return** result; 28. } 30. /\*\* 31. \* 写入缓存 32. \* @param key 33. \* @param offset 34. \* @return result 35. \*/ 36. **public** **boolean** getBit(String key, **long** offset) { 37. **boolean** result = **false**; 38. **try** { 39. ValueOperations<Serializable, Object> operations = redisTemplate.opsForValue(); 40. result = operations.getBit(key, offset); 41. } **catch** (Exception e) { 42. e.printStackTrace(); 43. } 44. **return** result; 45. } 47. /\*\* 48. \* 写入缓存 49. \* @param key 50. \* @param value 51. \* @return 52. \*/ 53. **public** **boolean** set(**final** String key, Object value) { 54. **boolean** result = **false**; 55. **try** { 56. ValueOperations<Serializable, Object> operations = redisTemplate.opsForValue(); 57. operations.set(key, value); 58. result = **true**; 59. } **catch** (Exception e) { 60. e.printStackTrace(); 61. } 62. **return** result; 63. } 65. /\*\* 66. \* 写入缓存设置时效时间 67. \* @param key 68. \* @param value 69. \* @return 70. \*/ 71. **public** **boolean** set(**final** String key, Object value, Long expireTime) { 72. **boolean** result = **false**; 73. **try** { 74. ValueOperations<Serializable, Object> operations = redisTemplate.opsForValue(); 75. operations.set(key, value); 76. redisTemplate.expire(key, expireTime, TimeUnit.SECONDS); 77. result = **true**; 78. } **catch** (Exception e) { 79. e.printStackTrace(); 80. } 81. **return** result; 82. } 84. /\*\* 85. \* 批量删除对应的value 86. \* @param keys 87. \*/ 88. **public** **void** remove(**final** String... keys) { 89. **for** (String key : keys) { 90. remove(key); 91. } 92. } 94. /\*\* 95. \* 删除对应的value 96. \* @param key 97. \*/ 98. **public** **void** remove(**final** String key) { 99. **if** (exists(key)) { 100. redisTemplate.delete(key); 101. } 102. } 104. /\*\* 105. \* 判断缓存中是否有对应的value 106. \* @param key 107. \* @return 108. \*/ 109. **public** **boolean** exists(**final** String key) { 110. **return** redisTemplate.hasKey(key); 111. } 113. /\*\* 114. \* 读取缓存 115. \* @param key 116. \* @return 117. \*/ 118. **public** Object get(**final** String key) { 119. Object result = **null**; 120. ValueOperations<Serializable, Object> operations = redisTemplate.opsForValue(); 121. result = operations.get(key); 122. **return** result; 123. } 125. /\*\* 126. \* 哈希 添加 127. \* @param key 128. \* @param hashKey 129. \* @param value 130. \*/ 131. **public** **void** hmSet(String key, Object hashKey, Object value) { 132. HashOperations<String, Object, Object> hash = redisTemplate.opsForHash(); 133. hash.put(key, hashKey, value); 134. } 136. /\*\* 137. \* 哈希获取数据 138. \* @param key 139. \* @param hashKey 140. \* @return 141. \*/ 142. **public** Object hmGet(String key, Object hashKey) { 143. HashOperations<String, Object, Object> hash = redisTemplate.opsForHash(); 144. **return** hash.get(key, hashKey); 145. } 147. /\*\* 148. \* 列表添加 149. \* @param k 150. \* @param v 151. \*/ 152. **public** **void** lPush(String k, Object v) { 153. ListOperations<String, Object> list = redisTemplate.opsForList(); 154. list.rightPush(k, v); 155. } 157. /\*\* 158. \* 列表获取 159. \* @param k 160. \* @param l 161. \* @param l1 162. \* @return 163. \*/ 164. **public** List<Object> lRange(String k, **long** l, **long** l1) { 165. ListOperations<String, Object> list = redisTemplate.opsForList(); 166. **return** list.range(k, l, l1); 167. } 169. /\*\* 170. \* 集合添加 171. \* @param key 172. \* @param value 173. \*/ 174. **public** **void** add(String key, Object value) { 175. SetOperations<String, Object> set = redisTemplate.opsForSet(); 176. set.add(key, value); 177. } 179. /\*\* 180. \* 集合获取 181. \* @param key 182. \* @return 183. \*/ 184. **public** Set<Object> setMembers(String key) { 185. SetOperations<String, Object> set = redisTemplate.opsForSet(); 186. **return** set.members(key); 187. } 189. /\*\* 190. \* 有序集合添加 191. \* @param key 192. \* @param value 193. \* @param scoure 194. \*/ 195. **public** **void** zAdd(String key, Object value, **double** scoure) { 196. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 197. zset.add(key, value, scoure); 198. } 200. /\*\* 201. \* 有序集合获取 202. \* @param key 203. \* @param scoure 204. \* @param scoure1 205. \* @return 206. \*/ 207. **public** Set<Object> rangeByScore(String key, **double** scoure, **double** scoure1) { 208. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 209. redisTemplate.opsForValue(); 210. **return** zset.rangeByScore(key, scoure, scoure1); 211. }  214. /\*\* 215. \* 第一次加载的时候将数据加载到 redis 中 216. \* @param name 217. \*/ 218. **public** **void** saveDataToRedis(String name) { 219. **double** index = Math.abs(name.hashCode() % size); 220. **long** indexLong = **new** Double(index).longValue(); 221. **boolean** availableUsers = setBit("availableUsers", indexLong, **true**); 222. } 224. /\*\* 225. \* 第一次加载的时候将数据加载到redis中 226. \* @param name 227. \* @return 228. \*/ 229. **public** **boolean** getDataToRedis(String name) { 230. **double** index = Math.abs(name.hashCode() % size); 231. **long** indexLong = **new** Double(index).longValue(); 232. **return** getBit("availableUsers", indexLong); 233. } 235. /\*\* 236. \* 有序集合获取排名 237. \* @param key   集合名称 238. \* @param value 值 239. \*/ 240. **public** Long zRank(String key, Object value) { 241. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 242. **return** zset.rank(key, value); 243. }  246. /\*\* 247. \* 有序集合获取排名 248. \* @param key 249. \*/ 250. **public** Set<ZSetOperations.TypedTuple<Object>> zRankWithScore(String key, **long** start, **long** end) { 251. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 252. Set<ZSetOperations.TypedTuple<Object>> ret = zset.rangeWithScores(key, start, end); 253. **return** ret; 254. } 256. /\*\* 257. \* 有序集合添加 258. \* @param key 259. \* @param value 260. \*/ 261. **public** Double zSetScore(String key, Object value) { 262. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 263. **return** zset.score(key, value); 264. }  267. /\*\* 268. \* 有序集合添加分数 269. \* @param key 270. \* @param value 271. \* @param scoure 272. \*/ 273. **public** **void** incrementScore(String key, Object value, **double** scoure) { 274. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 275. zset.incrementScore(key, value, scoure); 276. }  279. /\*\* 280. \* 有序集合获取排名 281. \* @param key 282. \*/ 283. **public** Set<ZSetOperations.TypedTuple<Object>> reverseZRankWithScore(String key, **long** start, **long** end) { 284. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 285. Set<ZSetOperations.TypedTuple<Object>> ret = zset.reverseRangeByScoreWithScores(key, start, end); 286. **return** ret; 287. } 289. /\*\* 290. \* 有序集合获取排名 291. \* @param key 292. \*/ 293. **public** Set<ZSetOperations.TypedTuple<Object>> reverseZRankWithRank(String key, **long** start, **long** end) { 294. ZSetOperations<String, Object> zset = redisTemplate.opsForZSet(); 295. Set<ZSetOperations.TypedTuple<Object>> ret = zset.reverseRangeWithScores(key, start, end); 296. **return** ret; 297. } 298. }   使用时只要将这个工具类注入到自己的Service或者Controller中即可。 IMG_256 Nginx 在windows上（非集群环境）对Nginx配置。   1. #user  nobody; 2. worker\_processes  1; 4. #error\_log  logs/error.log; 5. #error\_log  logs/error.log  notice; 6. #error\_log  logs/error.log  info; 8. #pid        logs/nginx.pid;  11. events { 12. worker\_connections  1024; 13. }  16. http { 17. include       mime.types; 18. default\_type  application/octet-stream; 20. #log\_format  main  '$remote\_addr - $remote\_user [$time\_local] "$request" ' 21. #                  '$status $body\_bytes\_sent "$http\_referer" ' 22. #                  '"$http\_user\_agent" "$http\_x\_forwarded\_for"'; 24. #access\_log  logs/access.log  main; 26. sendfile        on; 27. #tcp\_nopush     on; 29. #keepalive\_timeout  0; 30. keepalive\_timeout  65; 32. #gzip  on; 34. upstream backend{ 35. server      127.0.0.1:8009; 36. } 37. server { 38. listen       80; 39. server\_name  localhost; 41. #charset koi8-r; 43. #access\_log  logs/host.access.log  main; 45. location / { 46. root   html; 47. index  index.html index.htm; 48. try\_files $uri $uri/ /index.html; # 配置根路径请求时默认为加载/index.html 49. } 50. location /goods { 51. try\_files $uri $uri/ /index.html; # 将所有请求重定向到index.html 52. } 53. location /customers { 54. try\_files $uri $uri/ /index.html; # 将所有请求重定向到index.html 55. } 56. location /sales { 57. try\_files $uri $uri/ /index.html; # 将所有请求重定向到index.html 58. } 59. location /api { 60. proxy\_pass  http://backend; 61. proxy\_set\_header    Host    $host; 62. proxy\_set\_header    X-Real-IP    $remote\_addr; 63. add\_header 'Access-Control-Allow-Origin' '\*' always; 64. add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always; 65. add\_header 'Access-Control-Allow-Headers' 'Authorization,DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range' always; 66. if ($request\_method = 'OPTIONS') { 67. return 204; 68. } 69. } 70. } 71. }  集群搭建 修改hosts  首先使用ifconfig查询三台虚拟机的ip： IMG_256 然后修改三台虚拟机的hosts文件，方便后面从主机名到ip的映射。 IMG_257 安装JDK步骤省略 **安装MySQL** **安装** sudo apt install mysql-server   * 1  **验证** 一般安装后会自动运行，使用下面指令验证：  sudo systemctl status mysql   * 1   IMG_256 这代表mysql在正常运行。  接下来需要对mysql进行一些修改，使其可以远程连接。   1. 确定服务器上的防火墙没有阻止 3306 端口   由于我的虚拟机环境，防火墙没有开启，这里省略。   1. MySQL配置文件中设置本地IP/localhost绑定   IMG_256  在/etc/mysql/mysql.conf.d/mysqld.cnf文件中找到bind-address这一行，这里原本是127.0.0.1，代表只监听从本机来的连接，将其改为0.0.0.0，监听所有来源。   1. 创建一个远程账户，并授权 2. **CREATE** USER 'remote'@'%' IDENTIFIED **BY** '123456'; 3. **GRANT** ALL **PRIVILEGES** **ON** \*.\* **TO** 'remote'@'%' **WITH** **GRANT** **OPTION**; 4. FLUSH **PRIVILEGES**;   IMG_256   1. 验证一下   IMG_256 这里在宿主机使用Navicat来测试，确保可以远程连接。 Nginx**安装** sudo apt install nginx   * 1  **验证** 和mysql一样一般安装后会自动运行，使用下面指令验证：  sudo systemctl status nginx  IMG_256 **修改配置** 通过apt安装的nginx的配置文件都在/etc/nginx中。 IMG_256 conf.d中可以存放自定义的配置文件，nginx.conf中为全局的默认配置，做出以下修改：   1. 将user从www-data改为root 2. 将最后的include /etc/nginx/sites-enabled/\*;注释掉，它会将你自己定义的location / {}覆盖掉，去到欢迎页面。  **自定义配置** /etc/nginx/conf.d路径下新建一个以conf为扩展名的文件。 这里我直接贴出配置文件，具体其中内容不再赘述。   1. upstream backend{ 2. ip\_hash; 3. server  lab1:8080; 4. server  lab2:8080; 5. server  lab3:8080; 6. } 7. server { 8. listen      80; 9. server\_name localhost; 11. location / { 12. root    /home/lsxuan/site; 13. **index**  **index**.html **index**.htm; 14. try\_files $uri $uri/ /**index**.html; # 配置根路径请求时默认为加载/**index**.html 15. } 16. location /favicon.ico { 17. } 18. location /goods { 19. try\_files $uri $uri/ /**index**.html; # 将所有请求重定向到**index**.html 20. } 21. location /customers { 22. try\_files $uri $uri/ /**index**.html; # 将所有请求重定向到**index**.html 23. } 24. location /sales { 25. try\_files $uri $uri/ /**index**.html; # 将所有请求重定向到**index**.html 26. } 27. location /api { 28. proxy\_pass  http://backend; 29. proxy\_set\_header    Host    $host; 30. proxy\_set\_header    X-**Real**-IP    $remote\_addr; 31. add\_header 'Access-Control-Allow-Origin' '\*' always; 32. add\_header 'Access-Control-Allow-Methods' 'GET, POST, OPTIONS' always; 33. add\_header 'Access-Control-Allow-Headers' 'Authorization,DNT,User-Agent,X-Requested-With,If-Modified-Since,Cache-Control,Content-Type,Range' always; 34. if ($request\_method = 'OPTIONS') { 35. **return** 204; 36. } 37. } 38. }  **验证** IMG_256 在宿主机通过ip访问虚拟机，看看nginx是否正常。 **Redis****安装** sudo apt install redis-server   * 1  **验证** 同前两者一样一般安装后会自动运行，使用下面指令验证：  sudo systemctl status redis-server   * 1   IMG_256 **编辑配置文件** 在/etc/redis/redis.conf找到bind，将其从127.0.0.1 ::1修改为0.0.0.0 ::1，原理同mysql的修改。 IMG_256 **验证** IMG_256 在宿主机使用以下指令验证其是否可以远程访问：  redis-cli -h ${host/ip} ping 部署前端 前端使用react在webstorm中开发。  在webstorm的命令行中使用以下命令打包。  npm run build  运行完成后会生成以下文件。  IMG_256  在windows上将这些文件拷贝到以下目录： IMG_256 在ubuntu上将这些文件拷贝到以下目录：  IMG_257 这个路径都是在配置文件中可以自定义的。 比如，在windows中： IMG_258 在ubuntu上： IMG_259 **后端打包** 使用maven的package打包。 IMG_256 将这个生成的jar放到ubuntu的任意一个目录即可。 运行 前端放到nginx设置的目录下即可。  后端使用java运行。  java -jar ${jar\_name}  这样运行会使用jar包内部的配置文件，如果要指定springboot的配置文件可以使用-Dspring.config.location参数。比如  java -jar back-0.0.1-SNAPSHOT.jar -Dspring.config.location=./appication.yml  这部分内容也写在我的博客中：[关于Mybatis Generator，Nginx，Redis，Sharding-JDBC等的尝试与踩坑\_此号已废20的博客-CSDN博客](https://blog.csdn.net/qq_40364929/article/details/130892605)  2.2 每次改造的效果的具体体现（运行截图及文字等的论述）是什么？  Sharding-JDBC改造：        插入若干条数据，可以看到数据被分散到多个数据源中。 Redis 这里先介绍一下，代码中如何使用redis    这里只是一个简单的处理，从redis中取出所有的product，如果没有再从数据库中取出并存到redis中，若redis中已经缓存，则直接返回给客户端。  以下是redis中缓存的内容：   Nginx 因为ip\_hash对同一局域网中的请求都会转发到同一后端服务器（这是nginx的实现算法决定的），所以这里先将ip\_hash注释掉，采用默认的轮询策略。        可以看到nginx将请求平均分配到了各个后端服务器。  2.3完整测试文档内容（包括测试结果）： | | | | |
| 结对开发过程记录 | | | |  |
| **（1）角色切换与任务分工** | | | | |
| 表1-1结对开发角色与任务分工 | | | | |
| | 日期 | 时间(HH:MM - HH:MM) | 驾驶员角色 | 领航员角色 | 本段时间的任务 | | --- | --- | --- | --- | --- | |  |  |  |  |  | |  |  |  |  |  | |  |  |  |  |  |   【注意】该表格可自行增加更多的行。 | | | | |
| **（2）工作日志** | | | | |
| 由领航员负责记录，记录结对开发期间的遇到的问题、两人如何通过交流合作解决每个问题的。 | | | | |
| 表1-2 结对开发工作日志 | | | | |
| | 日期/时间 | 问题描述 | 最终解决方法 | 交流过程 | | --- | --- | --- | --- | |  |  |  |  | |  |  |  |  | |  |  |  |  |   【注意】该表格可自行增加更多的行 | | | | |
| **（3）结对开发工作现场照片** | | | | |
| 请其他同学帮助拍摄结对开发现场照片至少2张。 | | | | |
| 13071045_wLhN 1  结对开发现场照片1 结对开发现场照片2 | | | | |
| 实验总结 | | | |  |
| 【本实验的收获与不足，对事件系统架构风格、观测者模式、消息中间件等的理解】 | | | | |
| 教师评语 | | | |  |
|  | | | | |