Redis客户端在Spring Boot 中的实现

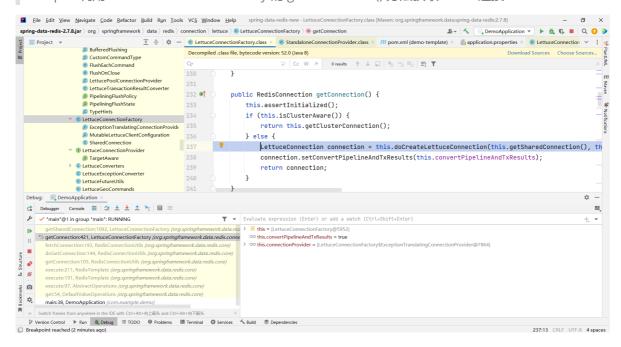
对比

序号	客户端	是否使用连接池	是否 shareNativeConnection=true	原理
1	Jedis	×	不支持	
2	Jedis	~	不支持	(Spring Data Redis 对于使用 Jedis 的默认配置)
3	Lettuce	×	✓	(Spring Data Redis 对于使用 Lettuce 的默认配置) 只和redis 服务端创建 一个 连接,对所有 Java中的redis调用进行多路共用 本地连接
4	Lettuce	~	✓	只和redis服务端创建一个连接,这个连接是从连接池取出来的,对所有Java中的redis调用进行共用本地连接。换句话说,这个连接池实际上没有任何作用,里面永远最多只保存一个连接
5	Lettuce	~	×	和redis服务端创建 多个 连接, Java中的redis调用会从连接池取 出来其中一个连接。这样的话, 连接池才真正发挥作用
6	Lettuce	×	×	和redis服务端创建 多个 连接,每次Java调用redis会新建一个连接,用完后释放

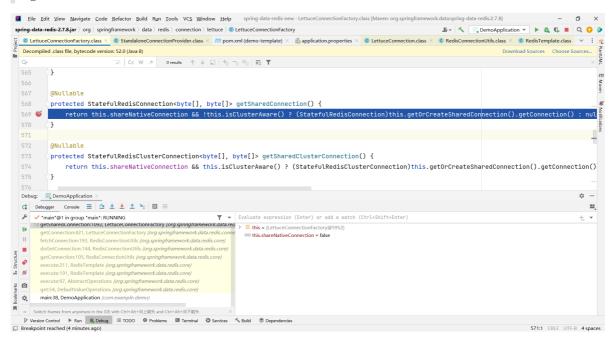
[•] 注:shareNativeConnection=true 这个参数是在配置 LettuceConnectionFactory 是否采用共享的本地连接

Lettuce源码单步分析(以上述序号4为例)

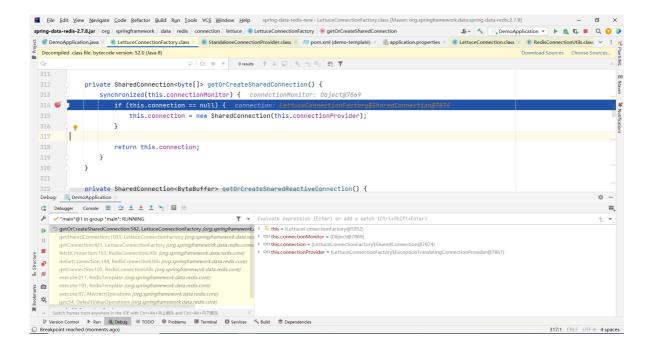
Step 1: 调用 LettuceConnectionFactory 的 getConnection() 方法获取 Redis 连接



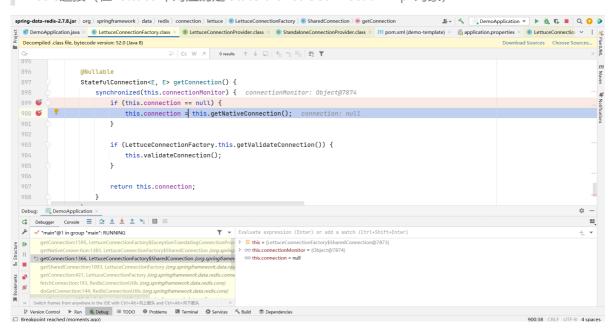
Step 2: 如果启用共享本地连接(默认行为),则会获取本地共享连接-getOrCreateSharedConnection()



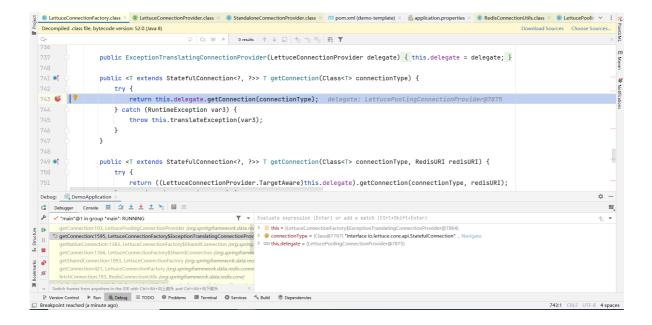
Step 3: getOrCreateSharedConnection() 通过线程安全的饿汉式的单例模式,返回 SharedConnection 对象。SharedConnection 为 LettuceConnectionFactory 的内部类。



Step 4: 对 Step 3 返回的 SharedConnection 对象,调用其 getConnection() 方法获取实际的物理 redis连接(在Lettuce 中对应的是 StatefuleRedisConnectionImpl 对象)



Step 5: 通过 LettuceConnectionProvider 类型的代理对象,继续获取连接



Step 6: 这里就是从连接池中借用连接的核心逻辑了。执行完下图中的57行,在redis-cli 中执行 "INFO Clients" 命令可以看到 Redis 服务端多了一个连接

```
class × 1 LettuceConnectionProvider.class × 2 StandaloneConnectionProvider.class × 3 StandaloneConnectionProvider.class × 1 pom.xml (demo-template) × 1 application.properties × 3 RedisConnectionUtils.class × 3 LettucePoolingConnectionProvider.class × 4 LettucePoolingConnectionProvid
                                          this.asyncPoolConfig = CommonsPool2ConfigConverter.bounded(this.poolConfig);
     48
                             public <T extends StatefulConnection<?, ?>> T getConnection(Class<T> connectionType) { connectionType: "interface io.lettuce.core.api.Stateful
                                   GenericObjectPool<StatefulConnection<?, ?>> pool = (GenericObjectPool)this.pools.computeIfAbsent(connectionType, (poolType) -> { pool:
      51
                                                 return ConnectionPoolSupport.createGenericObjectPool(() -> {
                                                             return this.connectionProvider.getConnection(connectionType); connectionType: "interface io.lettuce.core.api.StatefulConnection"
                                                 }, this.poolConfig, wrapConnections: false); poolConfig: "GenericObjectPoolConfig [lifo=true, fairness=false, maxWaitDuration=PT-0.0015,
                                  });
     56
                          StatefulConnection<?, ?> connection = (StatefulConnection)pool.borrowObject(); pool: " [maxTotal=5, blockWhenExhausted=tr
                                       this.poolRef.put(connection, pool);
                                                   return (StatefulConnection)connectionType.cast(connection);
                                                  throw new PoolException("Could not get a resource from the pool", var4);
    ☐ Debugger Console ☐ ☐ 立 🛨 🛨 🛨 🔭 🗏 🖽 🕾
                                                                                                                                                                                                                                                                                                                                                                                ₩,

✓ "main"@1 in group "main": RUNNING

                                                                                                                      GeonectionProvide > = this = (LettucePoolingConnectionProvider@7875)

GeonectionProv > @ connectionType = (Class@7767) *Interface iolettuce.core ction (org.spring.) > = pool = (ConnectionPoolSupportStandard.)
    ⇒ getConnection:103, LettucePoolingConnectionProvider (org.springframework.data.rec
   P Version Control ▶ Run 👢 Debug 🗏 TODO • Problems 🖾 Terminal • Services 🔨 Build 🕏 De
```

```
# Clients
connected_clients:2
cluster_connections:0
maxclients:10000
client_recent_max_input_buffer:8
client_recent_max_output_buffer:0
blocked_clients:0
tracking_clients:0
pubsub_clients:0
watching_clients:0
clients_in_timeout_table:0
total_watched_keys:0
total_blocking_keys:0
total_blocking_keys_on_nokey:0
127.0.0.1:6379>
```

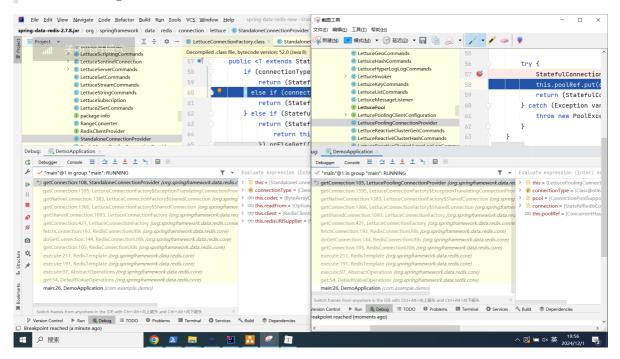
至此,第一个Lettuce连接就创建完成。**注意,之后的连接实际上不会再访问连接池了**。因为在 Step 4 中,this.connection 不为空,不会执行 900 行的 getNativeConnection(),直接返回已经创建好的连接对象。该连接对象存放在连接池中,也是池中的唯一元素。**所以这就是为什么说,在使用** sharedNativeConnection 的情况下,是否使用连接池效果是一样的。

其它源码分析

截图: Spring Data Redis 对于使用 Lettuce 的默认配置是 shareNativeConnection=true

```
© LettuceConnectionFactory.class × © StandaloneConnectionProvider.class × m pom.xml (demo-template) × 🚛 application.yml × 🐈 a
Decompiled .class file, bytecode version: 52.0 (Java 8)
                                                                                                         Downlo
                                                     ↑ ↓ □ | +<sub>II</sub> -<sub>II</sub> ⊠<sub>II</sub> | ≡<sub>I</sub> ▼
Q shareNativeConnection
                           × ⊋ Cc W .*
                   this((RedisStandaloneConfiguration)configuration, new MutableLettuceClientC
              }
              private LettuceConnectionFactory(LettuceClientConfiguration clientConfig) {
102 @ =
                   this.log = LogFactory.getLog(this.getClass());
                   this.validateConnection = false;
104
                   this. shareNativeConnection = true;
105
                   this.eagerInitialization = false;
                   this.connectionMonitor = new Object();
```

截图:展示了在shareNativeConnection=true 的情况下,不使用(左)和使用(右)连接池的代码逻辑分叉。对于使用连接池的情况,connectionProvider 是 LettucePoolingConnectionProvider



总结

- 1. Spring Data Redis 默认使用 Lettuce 作为客户端,在其外包了一层逻辑,能够共用本地连接(sharedNativeConnection)。这使得应用代码每次调用 redisTemplate 获取redis连接时,都是复用的同一个。Spring 的这个逻辑,与 Lettuce 的**多路复用(Multiflexing)原理**是相辅相成的:一个 LettuceConnectionFactory 只创建一个 sharedConnection;而一个 sharedConnection 对应一个物理连接 StatefuleRedisConnectionImpl;一个物理连接可以多路复用多个实际连接
- 2. 使用 Lettuce 作为客户端时是否需要使用连接池的问题,答案是不需要(原因如第1条)。而且就算是启用了连接池,在共用本地连接(sharedNativeConnection)的情况下,连接池里永远只有一个连接,没有任何意义
- 3. 如果**禁用**共用本地连接(sharedNativeConnection),那么连接池就有意义了,里面可以存储多个物理连接 StatefuleRedisConnectionImpl。不过实际上仔细想想,这也没有必要,因为一个物理连接 StatefuleRedisConnectionImpl 已经能多路复用所有的请求了
- 4. Redis 官方文档和 Lettuce 官方文档都不建议对 Lettuce 使用连接池
 - https://github.com/redis/lettuce/wiki/Connection-Pooling#asynchronous-connection-pooling

0

- 5. 再说 Jedis 客户端,默认是使用连接池的。而且 Jedis 必须使用连接池,不只是性能上的考虑,更重要的是 Jedis 连接(即Jedis对象)是线程不安全的,无法被多个请求共用,不像 Lettuce 有多路复用的特性
- 6. 使用 Lettuce 的时候,注意只有在 maven 中引入 commons-pool2 依赖,才能使用连接池。有了 该依赖 Spring autoconfiguration 会自动使用连接池,不必显式配置 pool.enable(不过我推荐显式配置,这样在忘记引用 commons-pool2 依赖时,程序会在启动时抛出异常)

附: 测试代码

主方法(在这里可以配置 Spring 的 LettuceConnectionFactory 的 shareNativeConnection):

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        ApplicationContext ctx = SpringApplication.run(DemoApplication.class,
args);
        StringRedisTemplate redisTemplate = ctx.getBean("stringRedisTemplate",
StringRedisTemplate.class);
        LettuceConnectionFactory connectionFactory =
ctx.getBean("redisConnectionFactory", LettuceConnectionFactory.class);
        connectionFactory.setShareNativeConnection(false); // hack in!
//
//
        for (int i=0; i<10; i++) {
//
            new Thread(() -> {
//
                String num = redisTemplate.opsForValue().get("num");
//
                System.out.println(Thread.currentThread().getName() + ": " +
num);
//
           }).start();
//
        String num = redisTemplate.opsForValue().get("num");
        System.out.println(Thread.currentThread().getName() + ": " + num);
        num = redisTemplate.opsForValue().get("num");
```

```
System.out.println(Thread.currentThread().getName() + ": " + num);
num = redisTemplate.opsForValue().get("num");
System.out.println(Thread.currentThread().getName() + ": " + num);
}
```

application.properties 配置:

```
spring.redis.lettuce.pool.enabled=true # 在有了common-pool2 的maven依赖时,可以不配置。但是推荐显式配置 spring.redis.lettuce.pool.minIdle=5 spring.redis.lettuce.pool.maxIdle=5 spring.redis.lettuce.pool.maxActive=5 logging.level.org.springframework.data.redis=DEBUG
```

pom文件:

```
<dependencies>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
     </dependency>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
     </dependency>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-redis</artifactId>
        <exclusions>
               <exclusion>-->
<!--
                  <!&ndash; Redis Client 1: Lettuce &ndash;&gt;-->
<!--
<!--
                  <artifactId>lettuce-core</artifactId>-->
                  <groupId>io.lettuce
<!--
<!--
               </exclusion>-->
        </exclusions>
     </dependency>
     <!-- Redis Client 2: Jedis -->
<!--
         <dependency>-->
<!--
            <groupId>redis.clients/groupId>-->
<!--
            <artifactId>jedis</artifactId>-->
         </dependency>-->
<!--
     <dependency>
        <groupId>org.apache.commons
         <artifactId>commons-pool2</artifactId>
     </dependency>
  </dependencies>
```