SpringBoot2之后，默认采用Lettuce作为redis的连接客户端。

lettuce采用netty连接redis server，实例可以在多个线程间共享，不存在线程不安全的情况，这样可以减少线程数量。当然，在特殊情况下，lettuce也可以使用多个实例。有点类似于NIO的模式。

此 demo 主要演示了 Spring Boot 如何整合 redis，操作redis中的数据，并使用redis缓存数据。连接池使用 Lettuce。

参考：https://github.com/xkcoding/spring-boot-demo/tree/master/spring-boot-demo-cache-redis

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>


    <artifactId>spring-boot-demo-cache-redis</artifactId>
    <version>1.0.0-SNAPSHOT</version>
    <packaging>jar</packaging>


    <name>spring-boot-demo-cache-redis</name>
    <description>Demo project for Spring Boot</description>


    <parent>
        <groupId>com.xkcoding</groupId>
        <artifactId>spring-boot-demo</artifactId>
        <version>1.0.0-SNAPSHOT</version>
    </parent>


    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>


    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
```

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>

        <!-- 对象池，使用redis时必须引入 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>

        <!-- 引入 jackson 对象json转换 -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-json</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
        </dependency>

        <dependency>
            <groupId>cn.hutool</groupId>
            <artifactId>hutool-all</artifactId>
        </dependency>

        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <build>
        <finalName>spring-boot-demo-cache-redis</finalName>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>


</project>
```

application.yml

```yaml
spring:
  redis:
    host: localhost
    password: ahhs2019
    # 连接超时时间（记得添加单位，Duration）
    timeout: 10000ms
    # Redis默认情况下有16个分片，这里配置具体使用的分片
    # database: 0
    lettuce:
      pool:
        # 连接池最大连接数（使用负值表示没有限制） 默认 8
        max-active: 8
        # 连接池最大阻塞等待时间（使用负值表示没有限制） 默认 -1
        max-wait: -1ms
        # 连接池中的最大空闲连接 默认 8
        max-idle: 8
        # 连接池中的最小空闲连接 默认 0
        min-idle: 0
  cache:
    # 一般来说是不用配置的，Spring Cache 会根据依赖的包自行装配
    type: redis
logging:
  level:
    com.xkcoding: debug
```

RedisConfig.java

```java
/**
 * <p>
 * redis配置
 * </p>
 *
 * @package: com.xkcoding.cache.redis.config
 * @description: redis配置
 * @author: yangkai.shen
 * @date: Created in 2018/11/15 16:41
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
@Configuration
@AutoConfigureAfter(RedisAutoConfiguration.class)
@EnableCaching
public class RedisConfig {


    /**
     * 默认情况下的模板只能支持RedisTemplate<String, String>，也就是只能存入字符串，因此支持序列化
     */
```

```
    @Bean
    public RedisTemplate<String, Serializable>
redisCacheTemplate(LettuceConnectionFactory redisConnectionFactory) {
        RedisTemplate<String, Serializable> template = new RedisTemplate<>();
        template.setKeySerializer(new StringRedisSerializer());
        template.setValueSerializer(new GenericJackson2JsonRedisSerializer());
        template.setConnectionFactory(redisConnectionFactory);
        return template;
    }


    /**
     * 配置使用注解的时候缓存配置，默认是序列化反序列化的形式，加上此配置则为 json 形式
     */
    @Bean
    public CacheManager cacheManager(RedisConnectionFactory factory) {
        // 配置序列化
        RedisCacheConfiguration config =
RedisCacheConfiguration.defaultCacheConfig();
        RedisCacheConfiguration redisCacheConfiguration =
config.serializeKeysWith(RedisSerializationContext.SerializationPair.fromSeriali
zer(new
StringRedisSerializer())).serializeValuesWith(RedisSerializationContext.Serializ
ationPair.fromSerializer(new GenericJackson2JsonRedisSerializer()));


        return
RedisCacheManager.builder(factory).cacheDefaults(redisCacheConfiguration).build(
);
    }
}
```

UserServiceImpl.java

```
/**
 * <p>
 * UserService
 * </p>
 *
 * @package: com.xkcoding.cache.redis.service.impl
 * @description: UserService
 * @author: yangkai.shen
 * @date: Created in 2018/11/15 16:45
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
@Service
@Slf4j
public class UserServiceImpl implements UserService {
    /**
     * 模拟数据库
     */
    private static final Map<Long, User> DATABASES = Maps.newConcurrentMap();
```

```
    /**
     * 初始化数据
     */
    static {
        DATABASES.put(1L, new User(1L, "user1"));
        DATABASES.put(2L, new User(2L, "user2"));
        DATABASES.put(3L, new User(3L, "user3"));
    }


    /**
     * 保存或修改用户
     *
     * @param user 用户对象
     * @return 操作结果
     */
    @CachePut(value = "user", key = "#user.id")
    @Override
    public User saveOrUpdate(User user) {
        DATABASES.put(user.getId(), user);
        log.info("保存用户【user】= {}", user);
        return user;
    }


    /**
     * 获取用户
     *
     * @param id key值
     * @return 返回结果
     */
    @Cacheable(value = "user", key = "#id")
    @Override
    public User get(Long id) {
        // 我们假设从数据库读取
        log.info("查询用户【id】= {}", id);
        return DATABASES.get(id);
    }


    /**
     * 删除
     *
     * @param id key值
     */
    @CacheEvict(value = "user", key = "#id")
    @Override
    public void delete(Long id) {
        DATABASES.remove(id);
        log.info("删除用户【id】= {}", id);
    }
}
```

RedisTest.java

主要测试使用 RedisTemplate 操作 Redis 中的数据：

- opsForValue：对应 String（字符串）
- opsForZSet：对应 ZSet（有序集合）
- opsForHash：对应 Hash（哈希）
- opsForList：对应 List（列表）
- opsForSet：对应 Set（集合）
- opsForGeo：** 对应 GEO（地理位置）

```
/**
 * <p>
 * Redis测试
 * </p>
 *
 * @package: com.xkcoding.cache.redis
 * @description: Redis测试
 * @author: yangkai.shen
 * @date: Created in 2018/11/15 17:17
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
@Slf4j
public class RedisTest extends SpringBootDemoCacheRedisApplicationTests {

    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    @Autowired
    private RedisTemplate<String, Serializable> redisCacheTemplate;

    /**
     * 测试 Redis 操作
     */
    @Test
    public void get() {
        // 测试线程安全，程序结束查看redis中count的值是否为1000
        ExecutorService executorService = Executors.newFixedThreadPool(1000);
        IntStream.range(0, 1000).forEach(i -> executorService.execute(() ->
stringRedisTemplate.opsForValue().increment("count", 1)));

        stringRedisTemplate.opsForValue().set("k1", "v1");
        String k1 = stringRedisTemplate.opsForValue().get("k1");
        log.debug("【k1】= {}", k1);

        // 以下演示整合，具体Redis命令可以参考官方文档
        String key = "xkcoding:user:1";
        redisCacheTemplate.opsForValue().set(key, new User(1L, "user1"));
        // 对应 String（字符串）
        User user = (User) redisCacheTemplate.opsForValue().get(key);
        log.debug("【user】= {}", user);
    }
}
```

UserServiceTest.java

主要测试使用Redis缓存是否起效

```
/**
 * <p>
 * Redis - 缓存测试
 * </p>
 *
 * @package: com.xkcoding.cache.redis.service
 * @description: Redis - 缓存测试
 * @author: yangkai.shen
 * @date: Created in 2018/11/15 16:53
 * @copyright: Copyright (c) 2018
 * @version: V1.0
 * @modified: yangkai.shen
 */
@Slf4j
public class UserServiceTest extends SpringBootDemoCacheRedisApplicationTests {
    @Autowired
    private UserService userService;

    /**
     * 获取两次，查看日志验证缓存
     */
    @Test
    public void getTwice() {
        // 模拟查询id为1的用户
        User user1 = userService.get(1L);
        log.debug("【user1】= {}", user1);

        // 再次查询
        User user2 = userService.get(1L);
        log.debug("【user2】= {}", user2);
        // 查看日志，查询用户【id】=    只打印一次日志，证明缓存生效
    }

    /**
     * 先存，再查询，查看日志验证缓存
     */
    @Test
    public void getAfterSave() {
        userService.saveOrUpdate(new User(4L, "测试中文"));

        User user = userService.get(4L);
        log.debug("【user】= {}", user);
        // 查看日志，只打印保存用户的日志，查询是未触发查询日志，因此缓存生效
    }

    /**
     * 测试删除，查看redis是否存在缓存数据
     */
    @Test
    public void deleteUser() {
```

```
        // 查询一次，使redis中存在缓存数据
        userService.get(1L);
        // 删除，查看redis是否存在缓存数据
        userService.delete(1L);
    }


}
```