链滴

springboot2.0+redis 整合及 redistemplate 简单使用

作者：someone44035

原文链接：https://ld246.com/article/1553501611992

来源网站：链滴

springboot2+redis 整合及redistemplate 简单使用

项目中涉及的所有代码均可在github中找到 [https://github.com/G-little/priest] (https://github.om/G-little/priest)

maven 依赖配置

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
    <version>2.1.3.RELEASE</version>
</dependency>
```

redis 配置

基于springboot项目结构的配置

```
* 单机

spring:
  cache:
    type: redis
  redis:
    host: 127.0.0.1
    port: 6379
    timeout: 0
    database: 0
    pool:
```

```
        max-active: 8
        max-wait: -1
        max-idle: 8
        min-idle: 0
```

* 集群

```
        spring:
            redis:
              cluster:
                nodes:
                  - 192.168.1.236:7001
                  - 192.168.1.236:7002
                max-redirects: 3  # 获取失败 最大重定向次数
              pool:
                max-active: 1000  # 连接池最大连接数（使用负值表示没有限制）
                max-idle: 10    # 连接池中的最大空闲连接
                max-wait: -1   # 连接池最大阻塞等待时间（使用负值表示没有限制）
                min-idle:  5    # 连接池中的最小空闲连接
              timeout: 6000  # 连接超时时长（毫秒）
```

非springboot结构spring 独立配置

redis properties 文件转javabean

```
package com.little.g.common.cache;


import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.PropertySource;
import org.springframework.stereotype.Component;

@Component
@EnableConfigurationProperties
@PropertySource(value = "classpath:/redis.properties",name = "redisProperties")
@ConfigurationProperties(prefix = "redis")
public class RedisProperties {

    public RedisProperties() {
    }

    private String nodes;

    private Integer commandTimeout;

    private Integer maxAttempts;

    private Integer maxRedirects;

    private Integer maxActive;
```

```
private Integer maxWait;

private Integer maxIdle;

private Integer minIdle;

private boolean testOnBorrow;

private String password;


public String getNodes() {
    return nodes;
}

public void setNodes(String nodes) {
    this.nodes = nodes;
}

public Integer getCommandTimeout() {
    return commandTimeout;
}

public void setCommandTimeout(Integer commandTimeout) {
    this.commandTimeout = commandTimeout;
}

public Integer getMaxAttempts() {
    return maxAttempts;
}

public void setMaxAttempts(Integer maxAttempts) {
    this.maxAttempts = maxAttempts;
}

public Integer getMaxRedirects() {
    return maxRedirects;
}

public void setMaxRedirects(Integer maxRedirects) {
    this.maxRedirects = maxRedirects;
}

public Integer getMaxActive() {
    return maxActive;
}

public void setMaxActive(Integer maxActive) {
    this.maxActive = maxActive;
}

public Integer getMaxWait() {
    return maxWait;
}
```

```
    public void setMaxWait(Integer maxWait) {
        this.maxWait = maxWait;
    }

    public Integer getMaxIdle() {
        return maxIdle;
    }

    public void setMaxIdle(Integer maxIdle) {
        this.maxIdle = maxIdle;
    }

    public Integer getMinIdle() {
        return minIdle;
    }

    public void setMinIdle(Integer minIdle) {
        this.minIdle = minIdle;
    }

    public boolean isTestOnBorrow() {
        return testOnBorrow;
    }

    public void setTestOnBorrow(boolean testOnBorrow) {
        this.testOnBorrow = testOnBorrow;
    }

    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    @Override
    public String toString() {
        return "RedisProperties{" +
                "nodes='" + nodes + '\'' +
                ", commandTimeout=" + commandTimeout +
                ", maxAttempts=" + maxAttempts +
                ", maxRedirects=" + maxRedirects +
                ", maxActive=" + maxActive +
                ", maxWait=" + maxWait +
                ", maxIdle=" + maxIdle +
                ", minIdle=" + minIdle +
                ", testOnBorrow=" + testOnBorrow +
                ", password='" + password + '\'' +
                '}';
    }
}
```

redis.properties 配置文件

```
# redis 集群地址 集群(ip:port)之间以， 分割
redis.nodes=${priest.redis.nodes}
# redis 集群密码
redis.password=${priest.redis.password}
# 超时时间
redis.command-timeout=15000
# 最大重试次数
redis.max-attempts=5
# 最大重定向次数
redis.max-redirects=3
# 最大活跃链接数
redis.max-active=16
# 连接池最大阻塞等待时间（使用负值表示没有限制）
redis.max-wait=-1
# 连接池中的最大空闲连接
redis.max-idle=8
# 连接池中的最小空闲连接
redis.min-idle=0
# 连接可靠测试
redis.test-on-borrow=true
```

RedisConfig 配置

Redistemplate ,connectionFactory, CacheManager,redisTemplate 相关配置

```
package com.little.g.common.cache;

import com.fasterxml.jackson.annotation.JsonAutoDetect;
import com.fasterxml.jackson.annotation.PropertyAccessor;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.apache.commons.lang.StringUtils;
import org.springframework.cache.CacheManager;
import org.springframework.cache.annotation.CachingConfigurerSupport;
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.cache.RedisCacheConfiguration;
import org.springframework.data.redis.cache.RedisCacheManager;
import org.springframework.data.redis.cache.RedisCacheWriter;
import org.springframework.data.redis.connection.RedisClusterConfiguration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.connection.RedisNode;
import org.springframework.data.redis.connection.RedisStandaloneConfiguration;
import org.springframework.data.redis.connection.jedis.JedisConnectionFactory;
import org.springframework.data.redis.core.*;
import org.springframework.data.redis.serializer.Jackson2JsonRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;
import org.springframework.util.CollectionUtils;
```

```
import javax.annotation.Resource;
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Configuration
@EnableCaching
public class RedisConfig extends CachingConfigurerSupport {


    @Resource
    private RedisProperties redisProperties;

    /**
     * 配置 Redis Cluster 信息
     */
    @Bean
    public RedisClusterConfiguration getJedisCluster() {
        RedisClusterConfiguration redisClusterConfiguration = new RedisClusterConfiguration();
        redisClusterConfiguration.setMaxRedirects(redisProperties.getMaxRedirects());

        List<RedisNode> nodeList = new ArrayList<>();

        String[] cNodes = redisProperties.getNodes().split(",");
        //分割出集群节点
        for(String node : cNodes) {
            String[] hp = node.split(":");
            nodeList.add(new RedisNode(hp[0], Integer.parseInt(hp[1])));
        }
        redisClusterConfiguration.setClusterNodes(nodeList);
        if(StringUtils.isNotEmpty(redisProperties.getPassword())){
            redisClusterConfiguration.setPassword(redisProperties.getPassword());
        }

        return redisClusterConfiguration;
    }



    @Bean
    public RedisConnectionFactory connectionFactory(RedisClusterConfiguration configuration)
{
        if(!CollectionUtils.isEmpty(configuration.getClusterNodes()) && configuration.getCluster
odes().size()>1) {
            return new JedisConnectionFactory(configuration);
        }
        Optional<RedisNode> node=configuration.getClusterNodes().stream().findFirst();
        RedisStandaloneConfiguration config = new RedisStandaloneConfiguration(node.get().g
tHost(), node.get().getPort());
        config.setPassword(configuration.getPassword());
        return new JedisConnectionFactory(config);
    }
```

```
    @Bean
    public CacheManager cacheManager(RedisConnectionFactory connectionFactory) {
        //初始化一个RedisCacheWriter
        RedisCacheWriter redisCacheWriter = RedisCacheWriter.nonLockingRedisCacheWriter(c
nnectionFactory);
        RedisCacheConfiguration defaultCacheConfig = RedisCacheConfiguration.defaultCache
onfig();
        //设置默认超过期时间是1天
        defaultCacheConfig.entryTtl(Duration.ofDays(1));
        //初始化RedisCacheManager
        RedisCacheManager cacheManager = new RedisCacheManager(redisCacheWriter, defaul
CacheConfig);
        return cacheManager;
    }



    /**
     * retemplate相关配置
     * @param factory
     * @return
     */
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory) {

        RedisTemplate<String, Object> template = new RedisTemplate<>();
        // 配置连接工厂
        template.setConnectionFactory(factory);

        //使用Jackson2JsonRedisSerializer来序列化和反序列化redis的value值（默认使用JDK的序列
方式）
        Jackson2JsonRedisSerializer jacksonSeial = new Jackson2JsonRedisSerializer(Object.class)


        ObjectMapper om = new ObjectMapper();
        // 指定要序列化的域，field,get和set,以及修饰符范围，ANY是都有包括private和public
        om.setVisibility(PropertyAccessor.ALL, JsonAutoDetect.Visibility.ANY);
        // 指定序列化输入的类型，类必须是非final修饰的，final修饰的类，比如String,Integer等会跑
异常
        om.enableDefaultTyping(ObjectMapper.DefaultTyping.NON_FINAL);
        jacksonSeial.setObjectMapper(om);

        // 值采用json序列化
        template.setValueSerializer(jacksonSeial);
        //使用StringRedisSerializer来序列化和反序列化redis的key值
        template.setKeySerializer(new StringRedisSerializer());

        // 设置hash key 和value序列化模式
        template.setHashKeySerializer(new StringRedisSerializer());
        template.setHashValueSerializer(jacksonSeial);
```

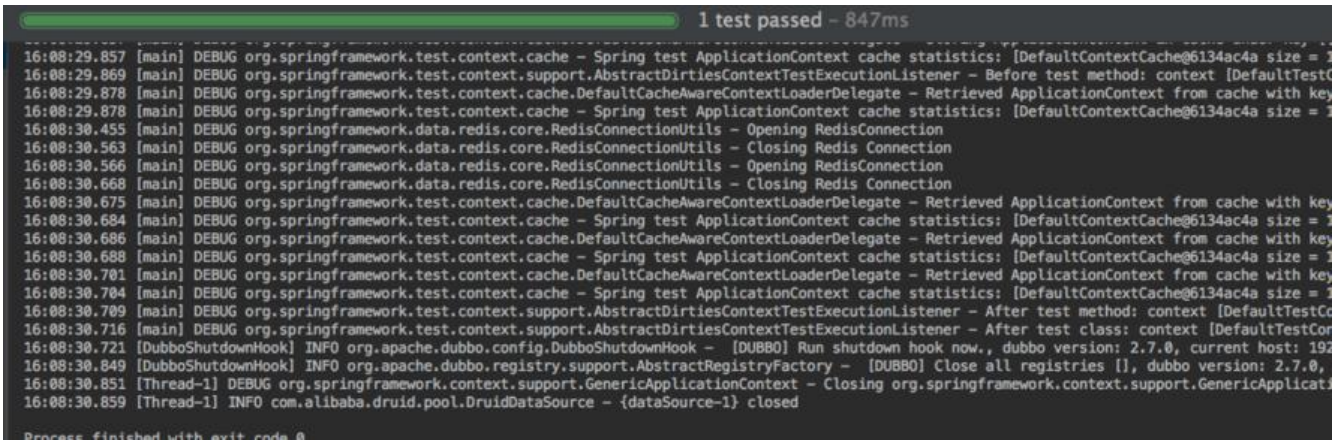
```
        template.afterPropertiesSet();

        return template;
    }

    /**
     * 对hash类型的数据操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public HashOperations<String, String, Object> hashOperations(RedisTemplate<String, Obj
ct> redisTemplate) {
        return redisTemplate.opsForHash();
    }

    /**
     * 对redis字符串类型数据操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public ValueOperations<String, Object> valueOperations(RedisTemplate<String, Object> r
disTemplate) {
        return redisTemplate.opsForValue();
    }

    /**
     * 对链表类型的数据操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public ListOperations<String, Object> listOperations(RedisTemplate<String, Object> redis
emplate) {
        return redisTemplate.opsForList();
    }

    /**
     * 对无序集合类型的数据操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public SetOperations<String, Object> setOperations(RedisTemplate<String, Object> redis
emplate) {
        return redisTemplate.opsForSet();
    }

    /**
```

```
     * 对有序集合类型的数据操作
     *
     * @param redisTemplate
     * @return
     */
    @Bean
    public ZSetOperations<String, Object> zSetOperations(RedisTemplate<String, Object> red
sTemplate) {
        return redisTemplate.opsForZSet();
    }
}
```

测试

```
package com.little.g.user;

import org.junit.Assert;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.data.redis.core.ValueOperations;
import org.springframework.test.context.ContextConfiguration;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

import javax.annotation.Resource;

/**
 * Created by lengligang on 2019/3/25.
 */

@RunWith(SpringJUnit4ClassRunner.class)
//根据自己的配置文件路径进行修改
@ContextConfiguration(locations = "classpath*:/META-INF/spring/*.xml")
public class RedisTest {

    @Resource
    private ValueOperations<String, String> valueOperations;

    @Test
    public void testRedisTemplate(){
        valueOperations.set("test","xxxx");
        String r=valueOperations.get("test");
        Assert.assertEquals(r,"xxxx");
    }

}
```

运行测试结果如下:



项目源代码 [https://github.com/G-little/priest] (https://github.com/G-little/priest)

spring redis官方文档 spring-data-redis