Spring Data Redis 是属于Spring Data 下的一个模块

作用是简化对于Redis的操作

Redis介绍

Redis是一个**开源**的使用ANSI C语言编写、遵守BSD协议、支持网络、可基于内存亦可持久化的日志型、Key-Value**数据库**，并提供多种语言的API。

它通常被称为数据结构**服务器**，因为值（value）可以是 字符串(String), 哈希(Hash), 列表(list), 集合(sets)和 有序集合(sorted sets)等类型。

随着Spring Boot支持的组件越来越多，技术也越来越成熟。在Spring Boot 2.X以后，对Redis的支持不再仅仅只是提供越来越强大的API，更是在底层，将Redis的默认客户端从 `Jedis` 改为了 `Lettuce` 。

#Lettuce和Jedis

`Lettuce` 和 `Jedis` 都是Java开发中，与Redis进行交互的中间件。 `Jedis` 在实现上是直连redis server，多线程环境下非线程安全，除非使用连接池，为每个Jedis实例增加物理连接。 `Lettuce` 基于Netty的连接实例（StatefulRedisConnection），可以在多个线程间并发访问，且线程安全，满足多线程环境下的并发访问，同时它是可伸缩的设计，一个连接实例不够的情况也可以按需增加连接实例。

添加依赖包

```
compile 'org.springframework.boot:spring-boot-starter-data-redis'
```

tips:在Spring Boot 2.X后， `Jedis` 的相关依赖已经被剔除，改为了 `Lettuce` 。

配置Redis

在 `application.yml` 中添加redis相关配置

```yaml
spring:
  redis:
    host: #地址
    port: #端口
    password: #密码
    timeout: # 连接超时时间（毫秒）
    lettuce:
      pool:
        max-active: #连接池最大连接数（使用负值表示没有限制） 默认 8
        max-wait: # 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-idle: # 连接池中的最大空闲连接 默认 8
        min-idle: # 连接池中的最小空闲连接 默认 0
```

XXXOperations

在Spring Boot中，对Redis的支持，其实主要是通过 `Spring Data Redis` 这个框架去完成的，它对Redis的支持，主要体现在框架中的XXXOperations，我们在实际开发中可以通过注入这些Operations来完成IO操作。下面简单的列举了几种常用的Operations

接口	描述
HashOperations	处理Hash结构的相关操作
ListOperations	处理List结构的相关操作
SetOperations	处理Set结构的相关操作
ValueOperations	处理String结构的相关操作
ZSetOperations	处理ZSet(或者 Sorted Set)结构的相关操作

2.yml文件配置

```
spring:

  datasource:
    username: root
    password: root
    url: jdbc:mysql://localhost:3306/vueblog?useUnicode=true&characterEncoding=utf-8&useSSL=true&serverTimezone=UTC
    driver-class-name: com.mysql.jdbc.Driver

  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
  redis:
    host: 127.0.0.1
    port: 6379
    password:
    timeout: 3600ms #超时时间
    jedis:
      pool:
        max-active: 8 #最大连接数
        max-idle: 8 #最大空闲连接 默认8
        max-wait: -1ms #默认-1 最大连接阻塞等待时间
        min-idle: 0 #最小空闲连接
```

1. 增加redis配置类 RedisConfig.java

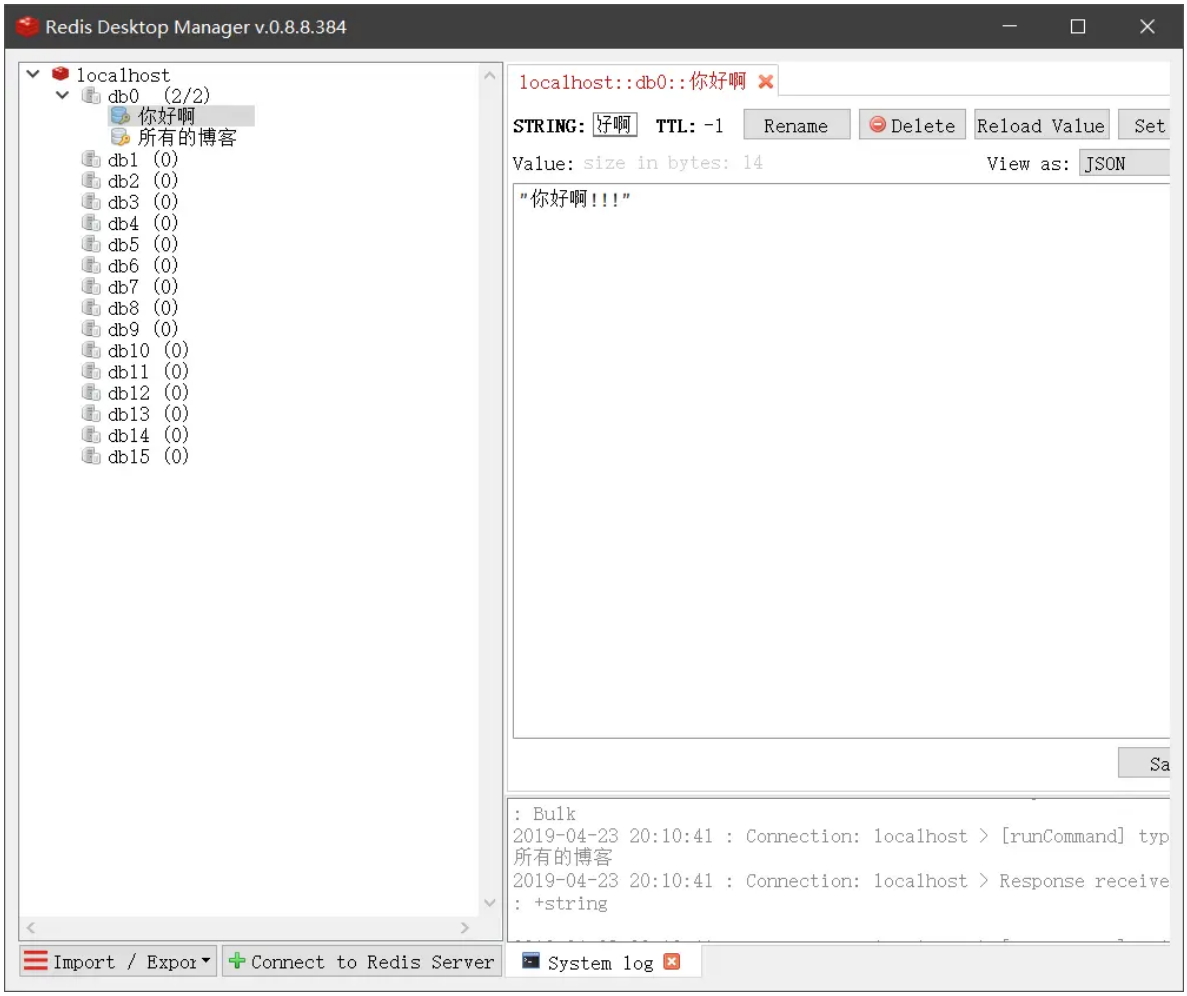
```
@Configuration
public class RedisConfig {
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory factory){
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        // 配置连接工厂
        template.setConnectionFactory(factory);
        //使用Jackson2JsonRedisSerializer来序列化和反序列化redis的value值（默认使用JDK的序列化方式）
        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等会跑出异常
        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;
    }

    public CacheManager cacheManager(RedisConnectionFactory redisConnectionFactory) {
// 生成一个默认配置，通过config对象即可对缓存进行自定义配置
        RedisCacheConfiguration config = RedisCacheConfiguration.defaultCacheConfig();
        config = config.entryTtl(Duration.ofMinutes(1))     // 设置缓存的默认过期时间，也是使用Duration设置
                .disableCachingNullValues();     // 不缓存空值
        // 设置一个初始化的缓存空间set集合
        Set<String> cacheNames =  new HashSet<>();
        cacheNames.add("timeGroup");
        cacheNames.add("user");
        // 对每个缓存空间应用不同的配置
        Map<String, RedisCacheConfiguration> configMap = new HashMap<>();
        configMap.put("timeGroup", config);
        configMap.put("user", config.entryTtl(Duration.ofSeconds(120)));
        RedisCacheManager cacheManager =
RedisCacheManager.builder(redisConnectionFactory)// 使用自定义的缓存配置初始化一个cacheManager
                .initialCacheNames(cacheNames)  // 注意这两句的调用顺序，一定要先调用该方法设置初始化的缓存名，再初始化相关的配置
                .withInitialCacheConfigurations(configMap)
                .build();
        return cacheManager;
    }
}
```

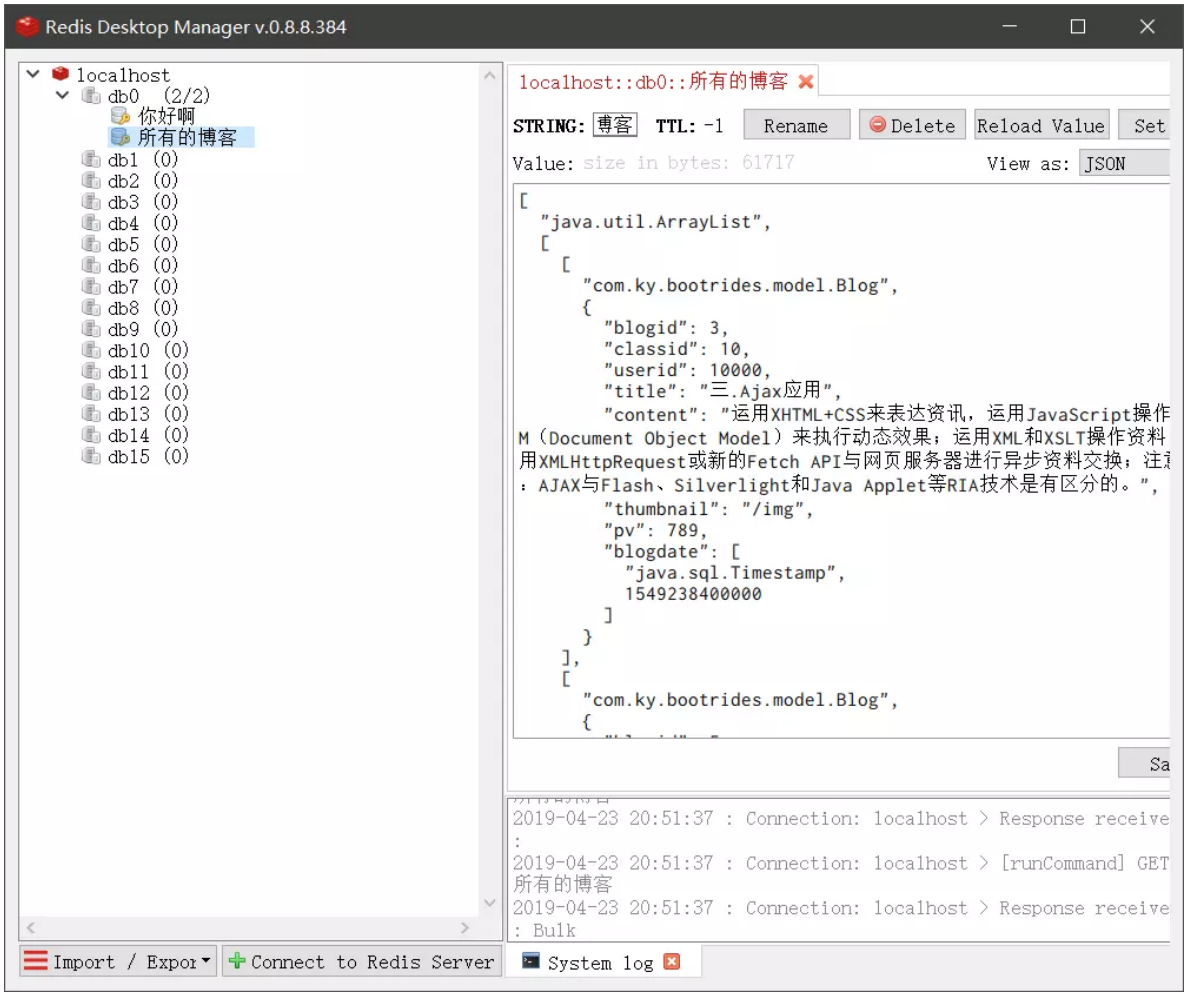
4.redisTemplate具体使用方法

```
redisTemplate.opsForValue().set("你好啊","你好啊!!!");
```



效果1.png

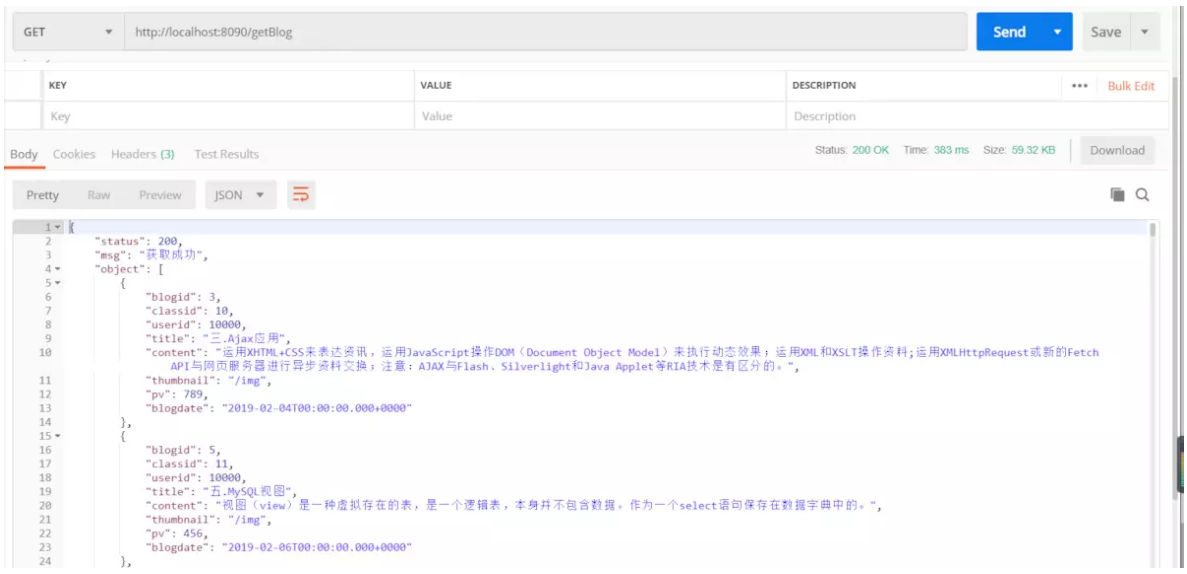
```
ValueOperations vo = redisTemplate.opsForValue();
        vo.set("所有的博客",all);
```



效果二.png

查询缓存中的 “所有的博客” 缓存

```
@RequestMapping("/getRedisBlog")
    public Result getredisBlog(){
        ValueOperations operations = redisTemplate.opsForValue();
        Object result = operations.get("所有的博客");
        return new Result(200,"获取成功",result);
    }
```



效果3.png

ValueOperations接口说明

这个接口的实现类为DefaultValueOperations，default这个类同时继承AbstractOperation,我们来看下这个类的构造函数：

```
DefaultValueOperations(RedisTemplate<K, V> template) {
super``(template);
}
```

这个类非公开的，需要传入template来构造。但是我们是无法访问的。不过不要急，在RedisTemplate中，已经提供了一个工厂方法:opsForValue()。这个方法会返回一个默认的操作类。另外，我们可以直接通过 `ValueOperations operations = redisTemplate.opsForValue();` 来进行注入。
opsForValue()集合使用说明 1). set(K key,V value) 新建缓存
`redisTemplate.opsForValue().set("key","value");` 2). get(Object key) 获取缓存
`edisTemplate.opsForValue().get("key");`