# Spring Boot 连接 Redis

## Redis的介绍

### 特点

为什么要使用Redis。难道就因为它是一个数据库，我们就应该使用它吗？以下是我认为的原因：

* Redis是一个高性能的内存数据库，可以高效地帮助我们管理存储在内存中的数据，且Redis存储和读取数据很快
* Redis是一个nosql数据库，可以存储非结构化数据，即不能存储在二维表中的数据
* Redis简单易学，学习成本低，且是开源的，获得了一众企业的信任和使用
* 跟Mybatis、JPA等ORM框架的一级缓存、二级缓存相比，Redis在缓存管理方面更高效
* Redis有着一系列的图形化管理工具，如Tiny RDM 、Navicat、VSCODE插件、DataGrip等

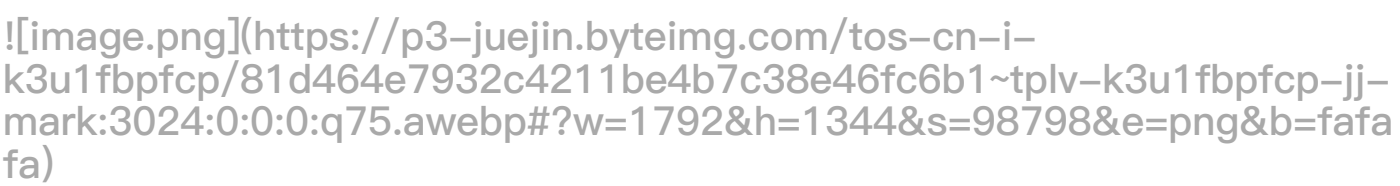
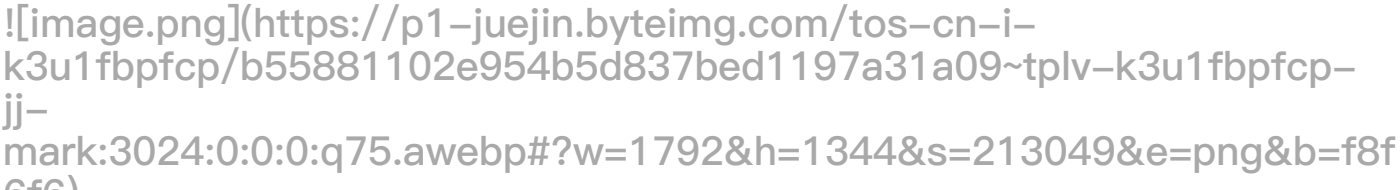
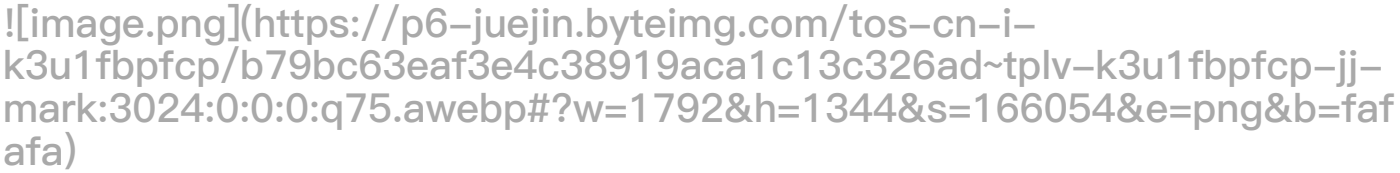
### 作为缓存层使用

当然了，如果想要使用Redis作为缓存层，那么请关闭自己ORM框架的一级缓存和二级缓存功能。我只介绍如何关闭Mybatis的一级缓存和二级缓存。

Mybatis的一级缓存是默认开启的，是无法彻底关闭的，二级缓存是默认关闭的。所以想要彻底关闭Mybatis的缓存功能，是不可能的。但是我们可以在application.yaml中进行如下配置：

```
mybatis:
  configuration:
    cache-enabled: false  #关闭二级缓存
    local-cache-scope: statement  #设置一级缓存处于statement范围, 即每
```

个SQL语句都默认不会使用一级缓存中的内容,每查询一次会清空一次一级缓存

```

经过上面配置，每次访问还是会产生缓存，但是也会刷新之前的缓存。所以缓存量降到了最低。

Redis图形界面工具
----------

Redis有着一系列的图形化管理工具，如Tiny RDM 、Navicat、VSCODE插件、DataGrip等，而我向大家推荐的是Tiny RDM。这是Github中一个新的热门项目，我认为它的界面十分好看，而且也有许多好用的功能，如下是界面截图。










大家可以直接在Github上搜Tiny RDM，当然也可以点下面我给的链接进去。

[Tiny RDM -- Github](http://cxyroad.com/ "https://github.com/tiny-craft/tiny-rdm")

项目配置
====
```

application.yaml
---

我创建了Spring Boot 3.2.5 项目，并在application.yaml中进行配置Spring Data Redis。Spring Data Redis是Spring 官方特地给Spring开发者整合出来的一个Spring组件，帮助我们统一了使用Redis的API。总所周知，Java的Redis客户端主要有两个，一个是Jedis，另一个是Lettuce。而通过使用Spring Data Redis，我们可以在application.yaml中简单配置一下，即可选择想要使用的客户端，并且API不会改变。

```
#服务运行的端口
server:
    port: 8083

spring:
    application:
        name: test

    data:
        redis:
            #redis服务所在的主机IP，我这里是连接虚拟机上的redis服务
            host: 192.168.11.10
            #redis服务的port端口
            port: 6379
            #redis服务的认证密码
            password: 123456
            #连接0号数据库
            database: 0
            #连接Redis的客户端选择lettuce
            lettuce:
                #配置lettuce的连接池
                pool:
                enabled: true
                max-active: 32
                max-idle: 16
                min-idle: 0
```

pom.xml
---

除了在application.yaml中选择一下Spring Data Redis组件的基本配置，还需

要在pom.xml中引入依赖，注意了，**Spring Boot 3.2.5 对JDK最低要求是JDK17，当然我们也可以选择JDK21**。但在这里，我选择JDK17，因为我不想使用虚拟线程，使用平台线程就够了。

```
    <!--设置父项目为Spring Boot 3.2.5，帮助我们转自己的项目成3.2.5-->
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.5</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <dependencies>

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

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

        <!--整合了Spring Boot 使用 Redis 时需要的基本依赖-->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>

        <!--想要使用Redis连接池，需要的依赖-->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>
    </dependencies>

    <!--设置自己项目的远程仓库，需要下载JAR包时项目会优先前往该仓库下载-->
    <repositories>
        <repository>
            <id>alimaven</id>
            <name>aliyun maven</name>
<url>http://maven.aliyun.com/nexus/content/groups/public/</url>
```

```
</repository>
    </repositories>
```

Redis工具类封装
==========

配置RedisTemplate和ObjectMapper
----------

Spring Data Redis主要提供了一个封装好的类 —— RedisTemplate。如果只是学习使用，那么不需要像我一样对RedisTemplate进行特殊配置。但是你如果不想存储一些特殊类型的数据时报错，我建议你还是要对RedisTemplate进行配置。在下面的代码中，我给RedisTemplate的key配置的是String类型，value也是配置String类型。这代表着我们存储键值对进Redis中时，key和value实际上都是字符串。所以，我们需要手动将value对应的对象先转成字符串，再存入Redis中。读取数据时，也需要我们手动将读取出的字符串转成对应的对象。实现转换的类是ObjectMapper，这是jackson提供的，属于Spring Boot自带的。我们还是需要进行手动配置ObjectMapper，防止它转换对象成字符串时出错。

```
import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.databind.ObjectMapper;
import com.fasterxml.jackson.databind.SerializationFeature;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import
org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.serializer.RedisSerializer;

import java.text.SimpleDateFormat;


@Configuration
public class RedisConfig {

    @Bean
    public RedisTemplate<String, String>
redisTemplate(RedisConnectionFactory factory) {
        RedisTemplate<String, String> template = new RedisTemplate<>();
        //设置连接工厂
```

```
        template.setConnectionFactory(factory);

        // 设置key和value的序列化规则
        template.setKeySerializer(RedisSerializer.string());
        template.setValueSerializer(RedisSerializer.string());

        // 设置hashKey和hashValue的序列化规则
        template.setHashKeySerializer(RedisSerializer.string());
        template.setHashValueSerializer(RedisSerializer.string());

        // 设置支持事物
        template.setEnableTransactionSupport(true);
        template.afterPropertiesSet();

        return template;
    }

    @Bean
    public ObjectMapper objectMapper() {
        ObjectMapper objectMapper = new ObjectMapper();
        //转成json字符串时标准化属性
        objectMapper.enable(SerializationFeature.INDENT_OUTPUT);

        //只有不是null的属性才能转成json
objectMapper.setSerializationInclusion(JsonInclude.Include.NON_EMPTY)
;

        //将json字符串转成对象时遇到未声明的类型属性和null, 不需要报错
objectMapper.configure(SerializationFeature.FAIL_ON_UNWRAPPED_TYP
E_IDENTIFIERS, false);
objectMapper.configure(SerializationFeature.FAIL_ON_EMPTY_BEANS, fa
lse);

        //配置正确的时间格式
objectMapper.configure(SerializationFeature.WRITE_DATES_AS_TIMEST
AMPS, false);
        objectMapper.setDateFormat(new SimpleDateFormat("yyyy-MM-
dd HH:mm:ss"));

        return objectMapper;
    }


}

```

Redis工具类代码

---

因为每次存储和读取都需要使用ObjectMapper，所以我对RedisTemplate<String,String>和ObjectMapper进行二次封装，可以减少代码冗余。下面的工具类，我并没有封装可以设置过期时间的方法，大家可以自己尝试进行二次封装。

```
import com.fasterxml.jackson.databind.ObjectMapper;
import jakarta.annotation.Resource;
import lombok.extern.slf4j.Slf4j;

import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
@Slf4j
public class RedisUtil {

    @Resource
    RedisTemplate<String, String> redisTemplate;

    @Resource
    ObjectMapper objectMapper;


    //判断是否存在key
    public boolean existsKey(String key) {
        return Boolean.TRUE.equals(redisTemplate.hasKey(key));
    }

    //判断某一hash是否存在某个key
    public boolean existsHashKey(String key, String hashKey) {
        return redisTemplate.opsForHash().hasKey(hashKey, key);
    }

    //存储一个Java对象进Redis中, 可以是单个Java对象, 也可以是集合对象
    public boolean storeObjects(String key, Object object) {
        try {
            String json = objectMapper.writeValueAsString(object);
            redisTemplate.opsForValue().set(key, json);
        } catch (Exception e) {
            log.error(e.getMessage());
```

```
            return false;
        }
        return true;
    }

    //获取单个Java对象, 不能是集合类
    public <T> T getObject(String key, Class<T> clazz) {
        String json = redisTemplate.opsForValue().get(key);
        if (json != null) {
            try {
                return objectMapper.readValue(json, clazz);
            } catch (Exception e) {
                log.error(e.getMessage());
            }
        }
        return null;
    }

    //获取一个List, 需要传入List的元素类型
    public <T> List<T> getList(String key, Class<T> clazz) {
        String json = redisTemplate.opsForValue().get(key);
        if (json != null) {
            try {
                return objectMapper.readValue(json,
objectMapper.getTypeFactory().constructCollectionType(List.class,
clazz));
            } catch (Exception e) {
                log.error(e.getMessage());
            }
        }
        return null;
    }

    //将对象存入一个hash中
    public boolean storeHash(String key, String hashKey, Object object) {
        try {
            String json = objectMapper.writeValueAsString(object);
            redisTemplate.opsForHash().put(key, hashKey, json);
        } catch (Exception e) {
            log.error(e.getMessage());
            return false;
        }
        return true;
    }

    //从hashKey中取出list对象
    public <T> List<T> getHashList(String key, String hashKey, Class<T>
clazz) {
```

```
        String json = (String) redisTemplate.opsForHash().get(key,
hashKey);
        if (json != null) {
            try {
                return objectMapper.readValue(json,
objectMapper.getTypeFactory().constructCollectionType(List.class,
clazz));
            } catch (Exception e) {
                log.error(e.getMessage());
            }
        }
        return null;
    }

    //从hashKey中取出非集合对象
    public <T> T getHash(String key, String hashKey, Class<T> clazz) {
        String json = (String) redisTemplate.opsForHash().get(key,
hashKey);
        if (json != null) {
            try {
                return objectMapper.readValue(json, clazz);
            } catch (Exception e) {
                log.error(e.getMessage());
            }
        }
        return null;
    }

}
```

测试
==

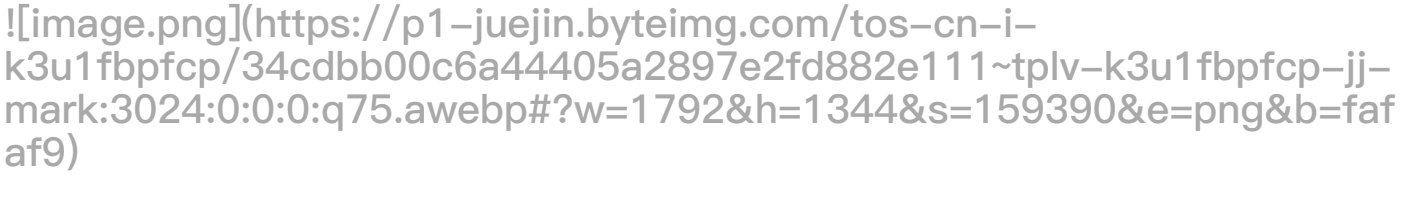
存储普通对象进Redis
----------

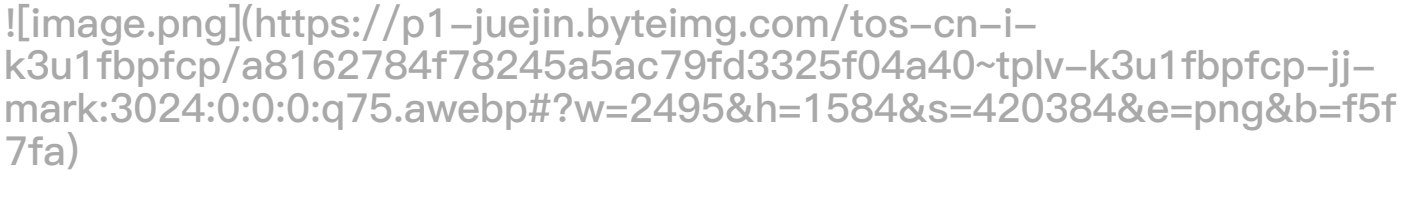
测试用例如下：

```
import com.fasterxml.jackson.core.JsonProcessingException;
import common.srtp.entity.User;
import jakarta.annotation.Resource;
```

```
import lombok.extern.slf4j.Slf4j;
import map.srtp.util.RedisUtil;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
@Slf4j
public class RedisTest {

    @Resource
    RedisUtil redisUtil;


    @Test
    public void writeObject() throws JsonProcessingException {
        User user = new User();
        user.setUserName(”Huonzy”);
        user.setOpenid(”123456”);
        user.setUserIcon(”/pictures/daeraon.png”);
        redisUtil.storeObjects(”user”, user);
    }

    @Test
    public void readObject() {
        User user = redisUtil.getObject(”user”, User.class);
        System.out.println(user);
    }

}
```

测试结果如下：



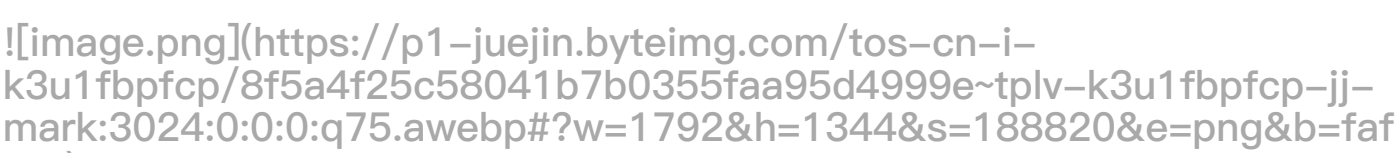
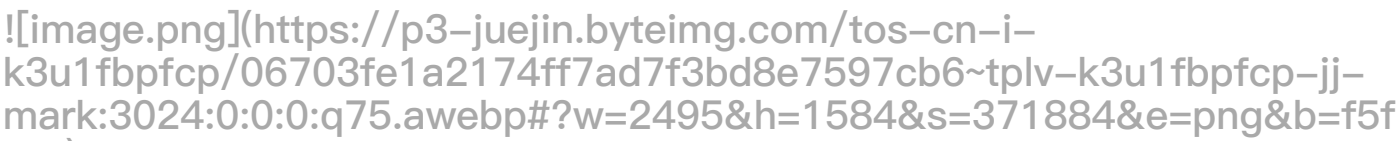


存储List对象进Redis

---

测试用例如下：

```
import com.fasterxml.jackson.core.JsonProcessingException;
import common.srtp.entity.User;
import jakarta.annotation.Resource;
import lombok.extern.slf4j.Slf4j;
import map.srtp.util.RedisUtil;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.ArrayList;
import java.util.List;

@SpringBootTest
@Slf4j
public class RedisTest {

    @Resource
    RedisUtil redisUtil;


    @Test
    public void writeObject() throws JsonProcessingException {
        List<User> users = new ArrayList<>();
        for (int i = 0; i < 3; i++) {
            User user = new User();
            user.setUserName("user" + i);
            user.setOpenid("openid" + i);
            user.setUserIcon("icon" + i);
            users.add(user);
        }
        redisUtil.storeObjects("users", users);
    }

    @Test
    public void readObject() {
        List<User> users = redisUtil.getList("users", User.class);
        System.out.println(users);
    }

}
```

```

测试结果如下：







在Redis的Hash中存储对象
---

测试用例如下：

```
import com.fasterxml.jackson.core.JsonProcessingException;
import common.srtp.entity.User;
import jakarta.annotation.Resource;
import lombok.extern.slf4j.Slf4j;
import map.srtp.util.RedisUtil;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

import java.util.ArrayList;
import java.util.List;

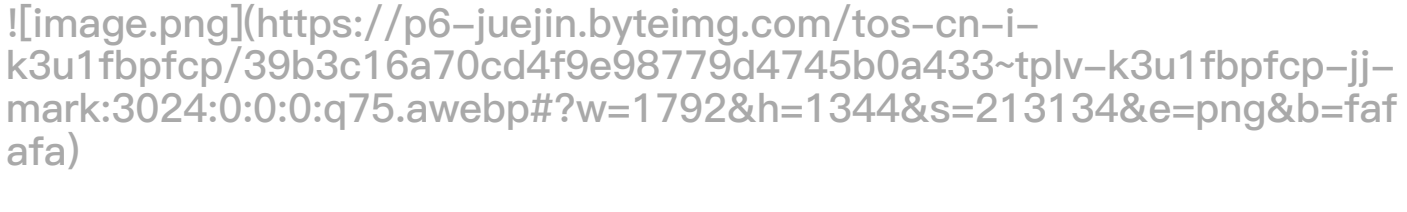
@SpringBootTest
@Slf4j
public class RedisTest {

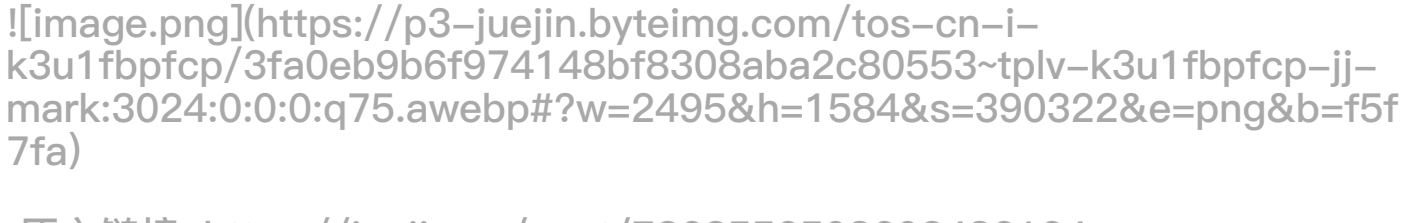
    @Resource
    RedisUtil redisUtil;


    @Test
    public void writeObject() throws JsonProcessingException {
        List<User> users = new ArrayList<>();
```
```

```
        for (int i = 0; i < 3; i++) {
            User user = new User();
            user.setUserName("user" + i);
            user.setOpenid("openid" + i);
            user.setUserIcon("icon" + i);
            users.add(user);
        }
        redisUtil.storeHash("paths", "users", users);
    }

    @Test
    public void readObject() {
        List<User> users = redisUtil.getHashList("paths", "users",
User.class);
        System.out.println(users);
    }

}
```

测试结果如下：





原文链接: https://juejin.cn/post/7363556508603482124