# Springboot(三)、缓存中间件 Redis 的使用

作者：TOJing
原文链接：https://ld246.com/article/1605520430747
来源网站：链滴

# Springboot(三)、缓存中间件Redis的使用

## 技术选用

- springboot
- guava
- redis
- postman
- AnotherRedisManager

## 整合Redis

- 首先启动redis服务，关于redis的安装和启动这里不再赘述。

```
[root@localhost redis-6.0.9]# sh /data/redisService.sh
1.start
2.status
3.stop
4.restart
please input cmd...
2
● redis.service - redis-server
   Loaded: loaded (/etc/systemd/system/redis.service; disabl
   Active: active (running) since 一 2020-11-16 15:04:54 CST
  Process: 8773 ExecStart=/usr/local/redis/bin/redis-server
 Main PID: 8774 (redis-server)
    Tasks: 4
   CGroup: /system.slice/redis.service
           └─8774 /usr/local/redis/bin/redis-server *:6379

11月 16 15:04:54 localhost.localdomain systemd[1]: Stopped r
11月 16 15:04:54 localhost.localdomain systemd[1]: Starting
11月 16 15:04:54 localhost.localdomain redis-server[8773]: 8
11月 16 15:04:54 localhost.localdomain redis-server[8773]: 8
11月 16 15:04:54 localhost.localdomain redis-server[8773]: 8
11月 16 15:04:54 localhost.localdomain systemd[1]: Started r
Hint: Some lines were ellipsized, use -l to show in full.
```

原文链接：Springboot(三)、缓存中间件 Redis 的使用

- 引入redis依赖以及guava工具包，

在父模块定义版本

```
<course.guava.version>18.0</course.guava.version>
```

在server模块引入依赖

```
<!-- 引入redis -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
            <version>${course.spring-boot.sersion}</version>
        </dependency>
        <dependency>
            <groupId>com.google.guava</groupId>
            <artifactId>guava</artifactId>
            <version>${course.guava.version}</version>
        </dependency>
```

- 在yaml文件中添加redis配置

```
spring:
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://192.168.119.110:3306/course?characterEncoding=UTF-8
    username: root
    password: 123456
  redis:
    port: 6379
    host: 192.168.119.110 # 这里填写自己的redis的ip和端口、密码
    password: 123456
mybatis:
  mapper-locations: classpath:sqlmap/**/*.xml
  type-aliases-package: com.to.jing.course.sdk
```

- 创建javaConfigure配置

新建包com.to.jing.course.server.config，在包下创建RedisConfig.java文件

```
package com.to.jing.course.server.config;

import org.springframework.beans.factory.annotation.Autowired;
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.core.StringRedisTemplate;
import org.springframework.data.redis.serializer.JdkSerializationRedisSerializer;
import org.springframework.data.redis.serializer.StringRedisSerializer;

@Configuration
public class RedisConfig {
    @Autowired
    private RedisConnectionFactory redisConnectionFactory;
```

```
    @Bean
    public RedisTemplate<String,Object> redisTemplate(){
        RedisTemplate<String,Object> redisTemplate = new RedisTemplate<>();
        redisTemplate.setConnectionFactory(redisConnectionFactory);
        //指定key序列化策略为String序列化，Value为JDK自带的序列化策略
        redisTemplate.setKeySerializer(new StringRedisSerializer());
        redisTemplate.setValueSerializer(new JdkSerializationRedisSerializer());
        //指定hashKey序列化策略为String序列化
        redisTemplate.setHashKeySerializer(new StringRedisSerializer());
        return redisTemplate;
    }
    @Bean
    public StringRedisTemplate stringRedisTemplate(){
        // 采用默认配置即可，后续有自定义注入配置时在此处添加即可
        StringRedisTemplate stringRedisTemplate = new StringRedisTemplate();
        stringRedisTemplate.setConnectionFactory(redisConnectionFactory);
        return stringRedisTemplate;
    }

}
```

这里我们主要引入两个对象RedisTemplate和StringRedisTemplate，指定缓存key与value的序列化略。StringRedisTemplate是对RedisTemplate的默认使用String序列化的实现，我们可以从StringRdisTemplate看到源码：

```
import ...

public class StringRedisTemplate extends RedisTemplate<String, String> {
    public StringRedisTemplate() {
        this.setKeySerializer(RedisSerializer.string());
        this.setValueSerializer(RedisSerializer.string());
        this.setHashKeySerializer(RedisSerializer.string());
        this.setHashValueSerializer(RedisSerializer.string());
    }

    public StringRedisTemplate(RedisConnectionFactory connectionFactory) {
        this();
        this.setConnectionFactory(connectionFactory);
        this.afterPropertiesSet();
    }

    protected RedisConnection preProcessConnection(RedisConnection connection, boolean existingConnection) {
        return new DefaultStringRedisConnection(connection);
    }
```

- 单元测试

在server模块下test/java目录下创建RedisTest.java文件

```
import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.JSONObject;
import com.to.jing.course.sdk.User;
import com.to.jing.course.server.AppServer;
import com.to.jing.course.service.UserService;
import org.junit.Test;
import org.junit.runner.RunWith;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.data.redis.core.ListOperations;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;


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

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest(classes = AppServer.class)
public class RedisTest {
    @Autowired
    private RedisTemplate<String,Object> redisTemplate;
    @Autowired
    private StringRedisTemplate stringRedisTemplate;
    @Autowired
    private UserService userService;

    @Test
    public void one(){
        final String key = "redis_test";
        final String content = "这是一个redis测试";
        redisTemplate.opsForValue().set(key,content);
        Object result = redisTemplate.opsForValue().get(key);
        System.out.println(result);
    }

    /**
     * 测试对象
     */
    @Test
    public void user(){
        User user = userService.findUserById(1);
        final String key = "test_user";
        stringRedisTemplate.opsForValue().set(key, JSON.toJSONString(user));
        User user1 = JSONObject.parseObject(stringRedisTemplate.opsForValue().get(key), User.c
ass);
        System.out.println(user1);
    }
    /**
     * 测试集合
     */
    @Test
    public void testList(){
        List<User> list = new ArrayList<>();
        list.add(userService.findUserById(1));
        final String key = "test_list";
        stringRedisTemplate.opsForValue().set(key,JSON.toJSONString(list));
        System.out.println(JSONObject.parseArray(stringRedisTemplate.opsForValue().get(key),U
er.class));
    }
```

```
    /**
     * 测试list
     */
    @Test
    public void testRedisList(){
        final String key = "test_redis_list";
        ListOperations<String, Object> stringObjectListOperations = redisTemplate.opsForList();
        for (int i = 0; i < 2 ; i++){
            User user = new User(i + 1,"hah","aa",23,true);
            stringObjectListOperations.leftPush(key,user);
        }
        Object res = stringObjectListOperations.rightPop(key);
        User temp;
        while (res != null){
            temp = (User) res;
            System.out.println(temp);
            res = stringObjectListOperations.rightPop(key);
        }
    }
    /**
     * list pushALL
     */
    @Test
    public void testRedisPushAll(){
        final String key = "test_push_all";
        List<User> users = new ArrayList<>();
        for (int i = 0 ;i< 2;i++){
            users.add(new User(i + 1,"hah","aa",23,true));
        }
        redisTemplate.opsForList().leftPushAll(key,users);
        List<Object> range = redisTemplate.opsForList().range(key, 0, -1);
        System.out.println(range);
    }
}
```

注意，这里会显示一些包引入错误，原因是之前引入springboot-test模块时配置错了，这里将原来的om文件的依赖替换一下。原来的为：

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-test</artifactId>
        <version>${course.spring-boot.sersion}</version>
    </dependency>
```

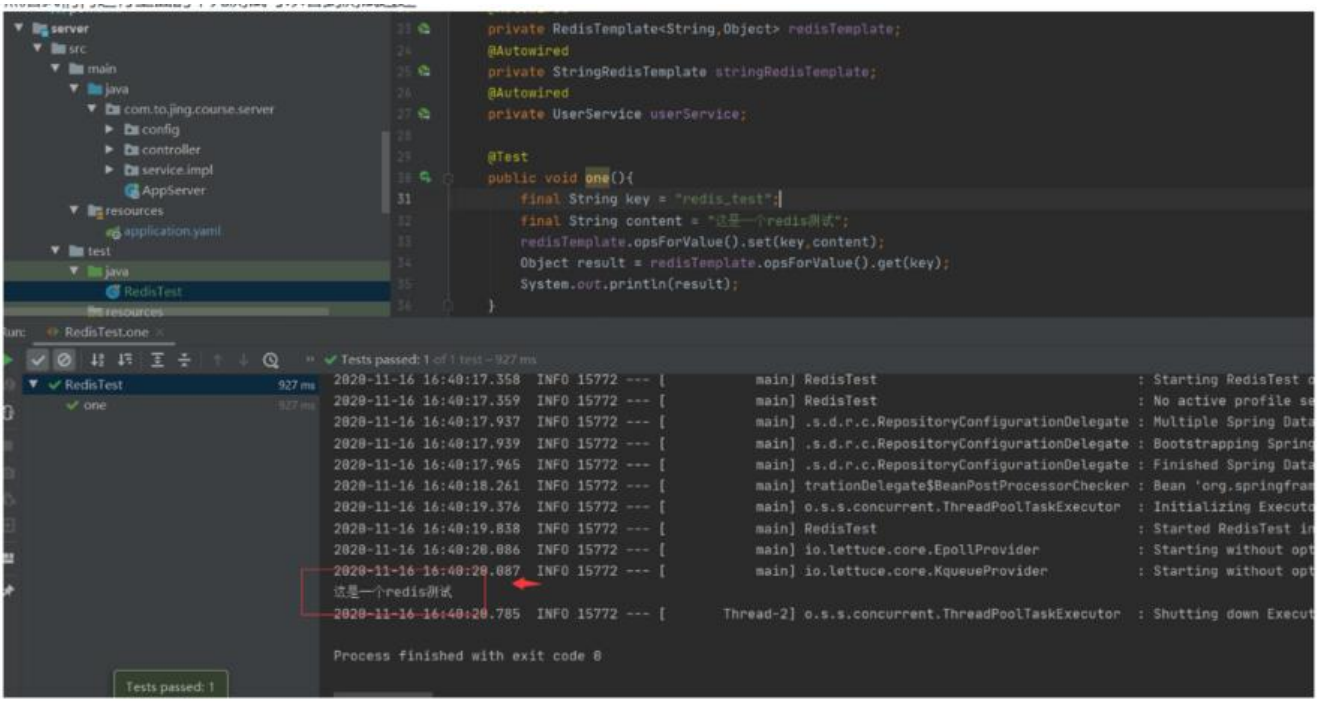
替换后为：

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <version>${course.spring-boot.sersion}</version>
        <scope>test</scope>
    </dependency>
```

替换后还是会显示错误，我们去User类下添加一些构造器,修改后代码如下，主要使用了lombok中的参构造@NoArgsConstructor和全参构造@AllArgsConstructor

```
package com.to.jing.course.sdk.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.io.Serializable;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class User implements Serializable {
    private Integer id;
    private String username;
    private String password;
    private Integer age;
    private Boolean disable;

    public static User Null(){
        User user = new User();
        user.setDisable(false);
        return user;
    }

}
```

然后我们再运行上面的单元测试可以看到测试通过




原文链接：Springboot(三)、缓存中间件 Redis 的使用

关于使用redisTemplate对redis中string,list,set,zset,hash数据结构的使用这里也不再详细说明

## 缓存穿透实战

在redis使用中常见的问题有缓存穿透、缓存雪崩、缓存击穿、数据一致性的问题。

● 缓存雪崩：指的是在某个时间点，缓存中的key集体过期失效，致使大量查询数据的请求都落在了据库上，导致数据库负载过高，压力暴增，甚至有可能压垮数据库。这种问题产生的原因其实主要是量的key再扣个时间点或者时间段过期失效，所以为了更好的避免这种情况的发生，一般的做法是为些key设置不同的，随机的TTL，从而错开缓存key的失效时间点，可以在某种程度上减轻数据库的查压力。

● 缓存击穿：指的是缓存中某个频繁被访问的key（可以被称为“热点key”）在不停地扛着前端地高发请求，当这个key突然在某个瞬间过期失效，持续地高并发访问请求就穿破缓存，直接访问数据库导致数据库压力在某一瞬间暴增。一般解决方案是不设置过期时间，但有些数据需要设置怎么办，可使用分布式锁加策略的方式。因为分布式锁只允许当前一个请求去查库，锁比较重，这种时候可以让它请求直接返回，或者拿着上一个时间段的热点数据进行返回。

● 缓存穿透：指的是所查询的数据不存在缓存中也不存在数据库中，这样每次请求过来都会经过缓存落在数据库上，缓存的效果就没了。如果前端频繁发起访问请求，恶意请求数据库中不存在的key，此时数据库中查询到的数据将永远为null，若被恶意攻击，发起“洪流”式查询，则很有可能会对数库造成极大的压力，甚至压垮“数据库”。一般的解决方法是将null结果也缓存在redis中，并设置过时间，或者用nginx对恶意ip进行黑白名单设置等等。

● 数据一致性：顾名思义就是缓存和数据库的数据保持一致。这种一般是对于需要更新数据来说，我应该是先更新数据库，然后再对redis键进行删除操作。这样前端进行查询的时候，自然会把新的数据进缓存里。

●

这里，我们通过代码实战一下缓存穿透。

首先，我们创建一个RedisService封装一下RedisTemplate和StringRedisTemplate，在包com.to.jin.course.server.service下。

```
package com.to.jing.course.server.service;

import com.alibaba.fastjson.JSON;
import com.google.common.base.Strings;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.data.redis.core.StringRedisTemplate;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;

import java.util.concurrent.TimeUnit;

@Service
public class RedisService {
    @Autowired
    private RedisTemplate<String,Object> redisTemplate;
    @Autowired
    private StringRedisTemplate stringRedisTemplate;

    public void setObject(String key,Object o){
        redisTemplate.opsForValue().set(key,o);
```

```
    }

    public void setString(String key,Object o){
        stringRedisTemplate.opsForValue().set(key, JSON.toJSONString(o));
    }
    public void setString(String key, Object o, Long time, TimeUnit timeUnit){
        stringRedisTemplate.opsForValue().set(key, JSON.toJSONString(o),time,timeUnit);
    }
    public Object getObject(String key){
        return redisTemplate.opsForValue().get(key);
    }

    public <T> T getString(String key , Class<T> clazz){
        String value = stringRedisTemplate.opsForValue().get(key);
        if (value == null || Strings.isNullOrEmpty(value)){
            return null;
        }
        return JSON.parseObject(value,clazz);
    }

    public Boolean hasKey(String key){
        return  redisTemplate.hasKey(key);
    }
}
```

创建redis键值前缀接口，主要定义一些唯一键值。在server模块新建包com.to.jing.course.server.cmmon，新建RedisPrefix接口。

```
package com.to.jing.course.server.common;

/**
 * redis键前缀
 */
public interface RedisPrefix {
    /**
     * 用户缓存键
     */
    String COURSE_CACHE_USER = "course_cache_user:";
}
```

然后在service模块userService接口中添加方法getUserInfo。

```
package com.to.jing.course.service;

import com.to.jing.course.sdk.domain.User;

public interface UserService {
    User findUserById(Integer id);
    User getUserInfo(Integer id);
}
```

server模块userServiceImpl对其实现，使用lombok的@Slf4j注解添加日志。

```
package com.to.jing.course.server.service.impl;

import com.to.jing.course.dao.UserDao;
import com.to.jing.course.sdk.domain.User;
import com.to.jing.course.server.common.RedisPrefix;
import com.to.jing.course.server.service.RedisService;
import com.to.jing.course.service.UserService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.Objects;
import java.util.concurrent.TimeUnit;

@Service
@Slf4j
public class UserServiceImpl implements UserService {
    @Autowired
    private UserDao userDao;
    @Autowired
    private RedisService redisService;

    @Override
    public User findUserById(Integer id) {
        return userDao.findUserById(id);
    }

    @Override
    public User getUserInfo(Integer id) {
        final String key = RedisPrefix.COURSE_CACHE_USER + id;
        User user = null;
        if (redisService.hasKey(key)){
            log.info("从缓存中获取用户信息");
            user = redisService.getString(key,User.class);
        }else {
            log.info("从数据库中获取用户信息");
            user = userDao.findUserById(id);
            if (Objects.isNull(user)){
                //用户不存在，缓存其空对象
                user = User.Null();
                user.setUsername("无效用户");
                redisService.setString(key,user,30L, TimeUnit.MINUTES);
            }else{
                redisService.setString(key,user);
            }
        }
        return user;
    }
}
```

其方法中的主要逻辑是先查看redis中是否存在键值，存在就获取redis中的数据返回，不存在就查库设置到缓存redis中，缓存穿透的解决就是遇到从数据库查询不到的数据也将空缓存到redis中并设置

定的过期时间，访问频繁地访问数据库。

在sdk模块创建Response统一一下响应数据的结构，在实际项目中还会创建响应码枚举类型。

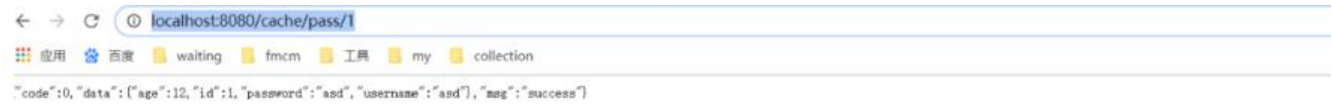
```
package com.to.jing.course.sdk.domain;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.Objects;

/**
 * 定义接口返回的数据格式 <br/>
 * 主要包括 code msg data
 */
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Response {
    private Integer code;
    private String msg;
    private Object data;

    public static Response SUCCESS(Object data){
        return new Response(0,"success",data);
    }

    public static Response SUCCESS(){
        Response response = new Response();
        response.setCode(0);
        response.setMsg("success");
        return response;
    }

    public void failed(String msg){
        this.code = -1;
        this.msg = msg;
        this.data = null;
    }
}
```

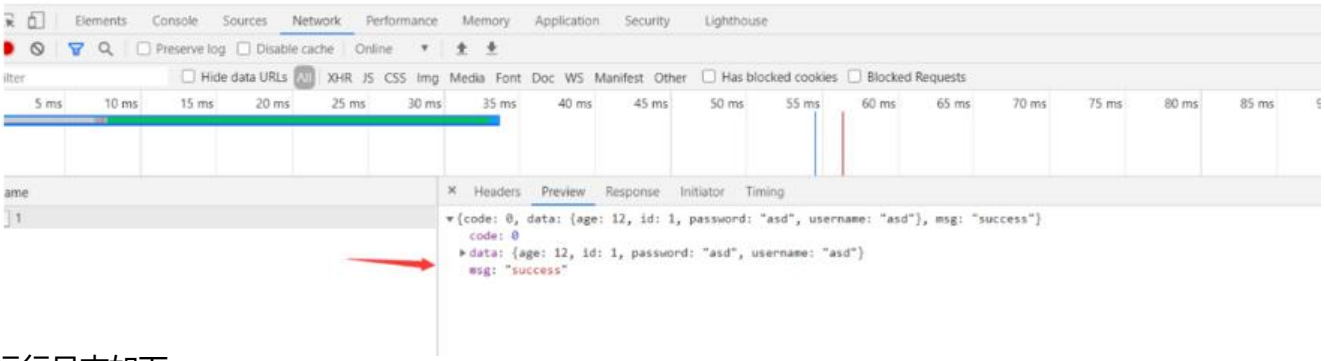
创建CachePassController.java,添加路由，这里使用了@PathVariable注解，可以直接从路由上获取数。

```
package com.to.jing.course.server.controller;

import com.alibaba.fastjson.JSON;
import com.alibaba.fastjson.serializer.SerializerFeature;
import com.to.jing.course.sdk.domain.Response;
import com.to.jing.course.service.UserService;
import lombok.extern.slf4j.Slf4j;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;
```

```
@RestController
@Slf4j
public class CachePassController {
    @Autowired
    private UserService userService;

    @RequestMapping(value = "/cache/pass/{id}" ,produces = "application/json;charset=UTF-8
)
    public String cachePass(@PathVariable("id") Integer id){

        //定义接口返回的数据格式
        Response response = Response.SUCCESS();
        try {
            response.setData(userService.getUserInfo(id));
        }catch (Exception e){
            response.failed("失败"+e.getMessage());
        }
        return JSON.toJSONString(response, SerializerFeature.BrowserCompatible);
    }
}
```

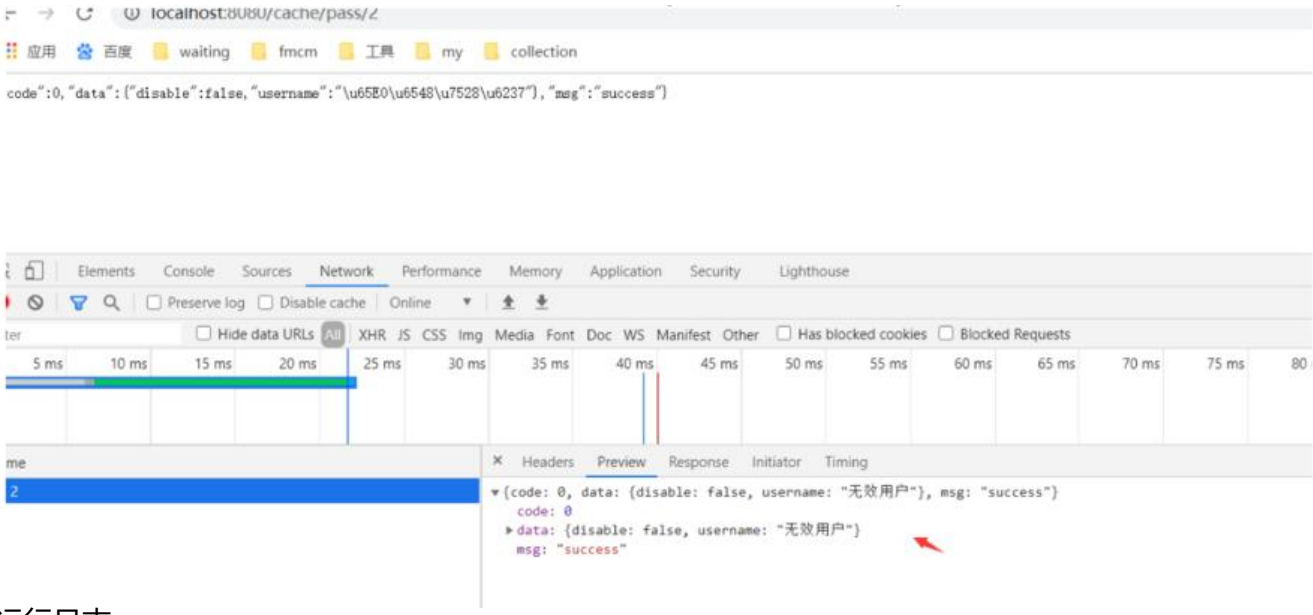
运行app，在浏览器中输入http://localhost:8080/cache/pass/1，可以看到结果如下图：



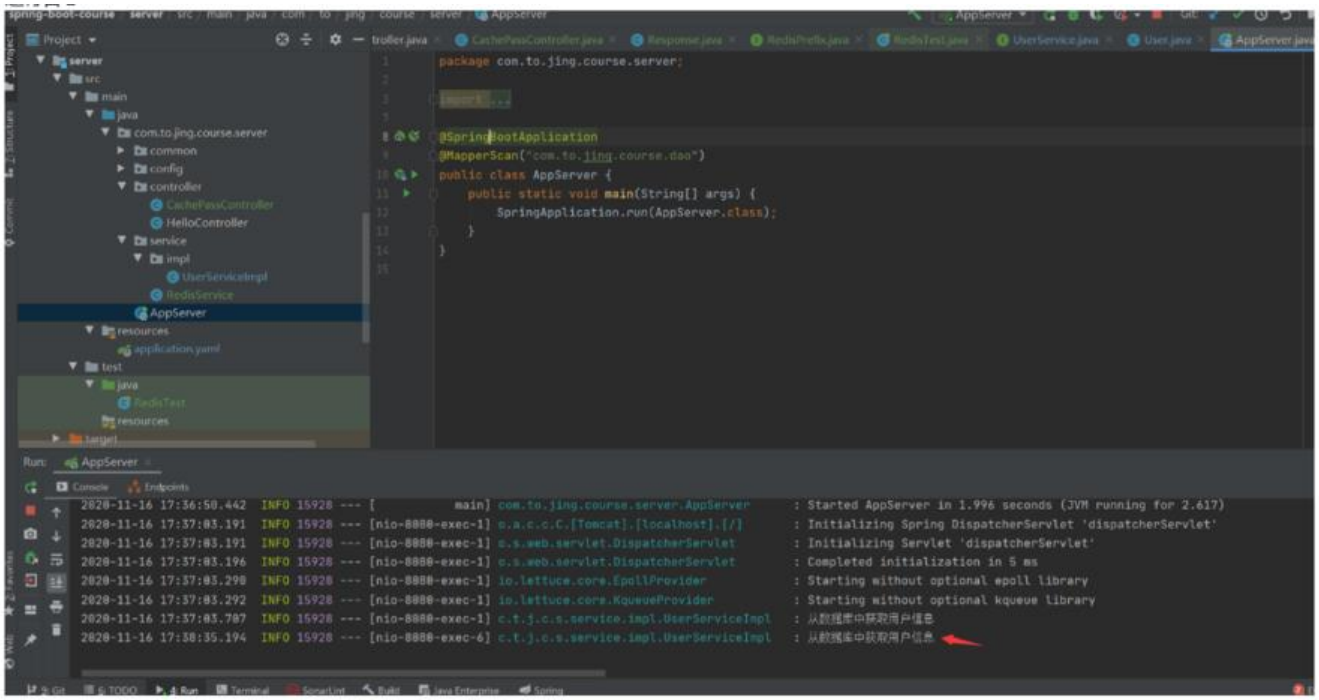




运行日志如下：

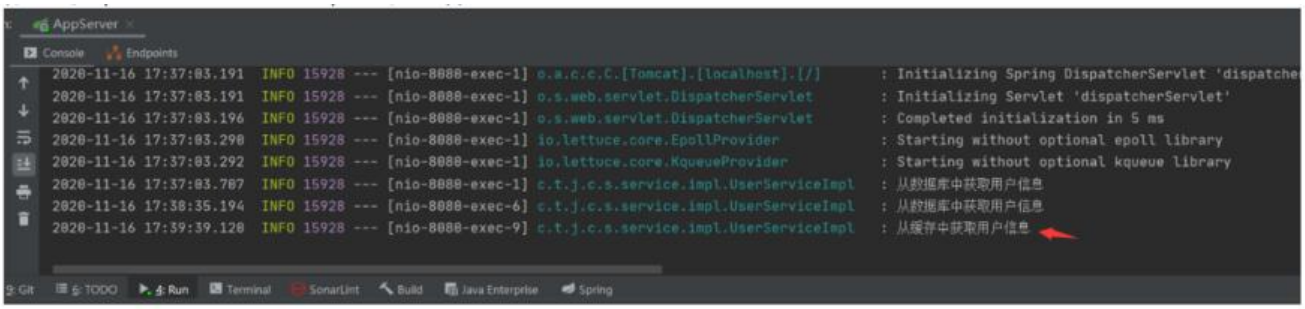

这是由于上一章节我们在表里创建了id为1的用户，下面我们访问一下id为2，http://localhost:8080/ache/pass/2



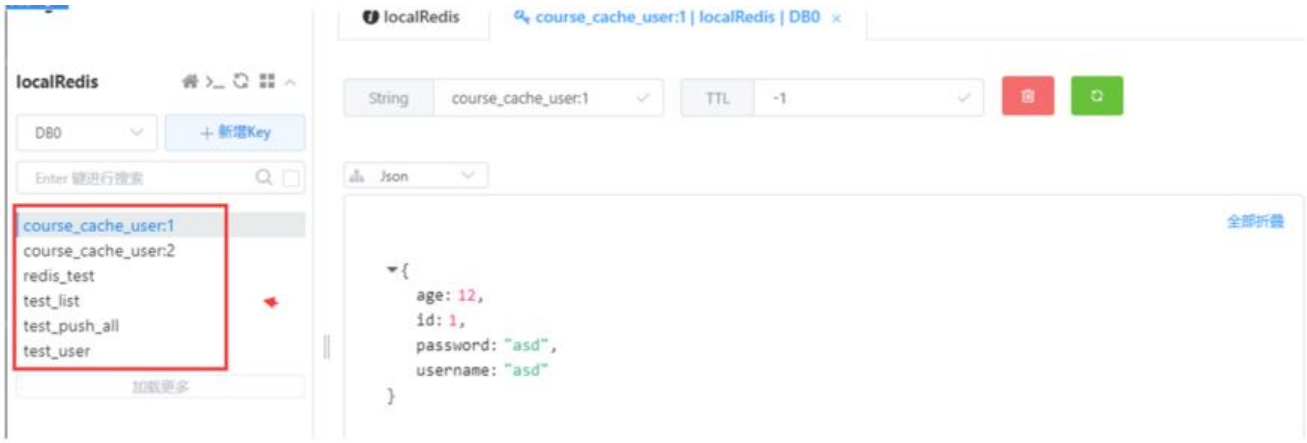

运行日志：




再次访问http://localhost:8080/cache/pass/2，运行日志为




从图中可以看到无效数据也被我们缓存到了redis中。这里更推荐小伙伴们使用postman测试接口，用anotherRedisMannager可以更直观地看到redis里的键值。




## 源码地址

https://github.com/ToJing/spring-boot-course tagV2.0

## 博客地址

http://m.loveplaycat.club/articles/2020/11/16/1605520428207.html

## 参考

- 书籍《基于Springboot实现Java分布式中间件开发入门与实战》