6-16 | 登录注册流程完善--手机号登录注册后台功能

手机号登录特点

1. 如果手机号没有注册过，就先注册，再登录
1. 如果手机号注册过，就直接登录

   userId，token

用户手机 RPC 服务的特点

读多写少场景，也需要考虑引入 Redis 缓存。

userId -> phone，phone -> userId , phone -> insert

表结构设计

我们需要创建一张用户手机号记录表，按照用户 id 进行分表存储，这里我们需要创建 100 张分表：

```
SQL
DELIMITER ;;
CREATE DEFINER=`root`@`%` PROCEDURE `create_t_user_phone_100`()
BEGIN

        DECLARE i INT;
        DECLARE table_name VARCHAR(30);
        DECLARE table_pre VARCHAR(30);
        DECLARE sql_text VARCHAR(3000);
        DECLARE table_body VARCHAR(2000);
        SET i=0;
        SET table_name='';

        SET sql_text='';
        SET table_body = ' (
  id bigint unsigned NOT NULL AUTO_INCREMENT COMMENT \'主键id\',
  phone varchar(200) CHARACTER SET utf8mb3 COLLATE utf8mb3_bin NOT
```

```
NULL DEFAULT \'\' COMMENT \'手机号\',
  user_id bigint DEFAULT -1 COMMENT \'用户id\',
  status tinyint DEFAULT -1 COMMENT \'状态(0无效，1有效)\',
  create_time datetime DEFAULT CURRENT_TIMESTAMP COMMENT \'创建时间
\',
  update_time datetime DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT \'更新时间\',
  PRIMARY KEY (id),
  UNIQUE KEY `udx_phone` (`phone`),
  KEY idx_user_id (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_bin;';

            WHILE i<100 DO
                IF i<10 THEN
                    SET table_name = CONCAT('t_user_phone_0',i);
                ELSE
                    SET table_name = CONCAT('t_user_phone_',i);
                END IF;

                SET sql_text=CONCAT('CREATE TABLE ',table_name,
table_body);
            SELECT sql_text;
            SET @sql_text=sql_text;
            PREPARE stmt FROM @sql_text;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;
            SET i=i+1;
        END WHILE;


    END;;
DELIMITER ;
```

用户手机表的 sql 结构如下：

```SQL
CREATE TABLE `t_user_phone` (
  `id` bigint unsigned NOT NULL COMMENT '主键id',
  `user_id` bigint NOT NULL DEFAULT '-1' COMMENT '用户id',
  `phone` varchar(200) CHARACTER SET utf8mb3 COLLATE utf8mb3_bin
NOT NULL COMMENT '手机号',
  `status` tinyint NOT NULL DEFAULT '-1' COMMENT '状态(0无效，1有
效)',
  `create_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP
```

```
COMMENT '创建时间',
  `update_time` datetime NOT NULL DEFAULT CURRENT_TIMESTAMP ON
UPDATE CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`),
  UNIQUE KEY `udx_phone` (`phone`),
  KEY `idx_user_id` (`user_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8mb3_bin;
```

用户手机 RPC 接口设计：

```
Java
package org.qiyu.live.user.interfaces;

import org.qiyu.live.user.dto.UserLoginDTO;
import org.qiyu.live.user.dto.UserPhoneDTO;

import java.util.List;


/**
 * @Author idea
 * @Date: Created in 14:01 2023/6/10
 * @Description
 */
public interface IUserPhoneRpc {

    /**
     * 手机号登录
     *
     * @param phone
     * @return
     */
    UserLoginDTO login(String phone);

    /**
     * 更具用户 id 查询手机信息
```

```
     *
     * @param userId
     * @return
     */
    List<UserPhoneDTO> queryByUserId(Long userId);

    /**
     * 根据手机号查询
     *
     * @param phone
     * @return
     */
    UserPhoneDTO queryByPhone(String phone);
}
```

用户手机 RPC 实现类

```
Java
package org.qiyu.live.user.provider.rpc;

import jakarta.annotation.Resource;
import org.apache.catalina.User;
import org.apache.dubbo.config.annotation.DubboReference;
import org.apache.dubbo.config.annotation.DubboService;
import
org.idea.qiyu.live.framework.redis.starter.key.UserProviderCacheKe
yBuilder;
import org.qiyu.live.id.generate.enums.IdTypeEnum;
import org.qiyu.live.id.generate.interfaces.IdGenerateRpc;
import org.qiyu.live.user.dto.UserDTO;
import org.qiyu.live.user.dto.UserPhoneDTO;
import org.qiyu.live.user.dto.UserLoginDTO;
import org.qiyu.live.user.interfaces.IUserPhoneRpc;
import org.qiyu.live.user.provider.service.IUserPhoneService;
import org.qiyu.live.user.provider.service.IUserService;
import org.qiyu.live.user.provider.service.bo.UserRegisterBO;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.util.CollectionUtils;

import java.util.*;
import java.util.concurrent.TimeUnit;
```

```
/**
 * @Author idea
 * @Date: Created in 19:58 2023/6/10
 * @Description
 */
@DubboService
public class UserPhoneRpcImpl implements IUserPhoneRpc {

    private static final Logger LOGGER =
LoggerFactory.getLogger(UserPhoneRpcImpl.class);

    @Resource
    private IUserPhoneService userPhoneService;
    @Resource
    private IUserService userService;
    @DubboReference
    private IdGenerateRpc idGenerateRpc;
    @Resource
    private RedisTemplate<String, UserPhoneDTO> redisTemplate;
    @Resource
    private RedisTemplate<String, Object>
stringObjectRedisTemplate;
    @Resource
    private UserProviderCacheKeyBuilder cacheKeyBuilder;

    @Override
    public UserLoginDTO login(String phone) {
        if (phone == null) {
            return UserLoginDTO.loginError("手机号不能为空");
        }
        UserPhoneDTO userPhoneDTO = this.queryByPhone(phone);
        if (userPhoneDTO != null) {
            LOGGER.error("phone is {} 已经被注册,可以直接登录",
phone);
            return
UserLoginDTO.loginSuccess(userPhoneDTO.getUserId(),
userPhoneDTO.getPhone(),
createAndSaveToken(userPhoneDTO.getUserId()));
        }
        //进行注册操作
        UserRegisterBO userRegisterBO = registerUser(phone);
        LOGGER.error("userPhoneDTO is {} 注册成功，进行登录",
userPhoneDTO);
```

```
        return
UserLoginDTO.loginSuccess(userRegisterBO.getUserId(), phone,
createAndSaveToken(userRegisterBO.getUserId()));
    }

    private String createAndSaveToken(Long userId) {
        String token = UUID.randomUUID().toString();
        String tokenKey =
cacheKeyBuilder.buildUserLoginTokenKey(userId);
        stringObjectRedisTemplate.opsForValue().set(tokenKey,
tokenKey, 30, TimeUnit.DAYS);
        return token;
    }

    private UserRegisterBO registerUser(String phone) {
        Long newUserId =
idGenerateRpc.getUnSeqId(IdTypeEnum.USER_ID.getCode());
        //将手机号右移 2 位，然后取末尾两位数字
        UserDTO userDTO = new UserDTO();
        userDTO.setUserId(newUserId);
        userDTO.setNickName("旗鱼用户-" + userDTO.getUserId());
        userService.insertOne(userDTO);
        userPhoneService.insert(phone, newUserId);
        UserRegisterBO userRegisterBO = new UserRegisterBO();
        userRegisterBO.setPhone(phone);
        userRegisterBO.setUserId(newUserId);
        return userRegisterBO;
    }


    @Override
    public List<UserPhoneDTO> queryByUserId(Long userId) {
        String redisKey =
cacheKeyBuilder.buildUserPhoneListKey(userId);
        List<UserPhoneDTO> userPhoneCacheList =
redisTemplate.opsForList().range(redisKey, 0, -1);
        if (!CollectionUtils.isEmpty(userPhoneCacheList)) {
            //可能是缓存的空值
            if (userPhoneCacheList.get(0).getUserId() == null) {
                return Collections.emptyList();
            }
            return userPhoneCacheList;
        }
        userPhoneCacheList =
```

```
userPhoneService.queryByUserId(userId);
        int expireTime = 30;
        if (CollectionUtils.isEmpty(userPhoneCacheList)) {
            //防止缓存击穿
            userPhoneCacheList = Arrays.asList(new
UserPhoneDTO());
            expireTime = 5;
        }
        redisTemplate.opsForList().leftPushAll(redisKey,
userPhoneCacheList);
        redisTemplate.expire(redisKey, expireTime,
TimeUnit.MINUTES);
        return userPhoneCacheList;
    }

    @Override
    public UserPhoneDTO queryByPhone(String phone) {
        String redisKey =
cacheKeyBuilder.buildUserPhoneObjKey(phone);
        UserPhoneDTO userPhoneDTO =
redisTemplate.opsForValue().get(redisKey);
        if (userPhoneDTO != null && userPhoneDTO.getUserId() !=
null) {
            return userPhoneDTO;
        }
        if (userPhoneDTO != null && userPhoneDTO.getUserId() ==
null) {
            return null;
        }
        userPhoneDTO = userPhoneService.queryByPhone(phone);
        if (userPhoneDTO != null) {
            redisTemplate.opsForValue().set(redisKey,
userPhoneDTO, 30, TimeUnit.MINUTES);
            return userPhoneDTO;
        }
        //缓存一个空对象，防止缓存击穿
        redisTemplate.opsForValue().set(redisKey, new
UserPhoneDTO(), 1, TimeUnit.MINUTES);
        return null;
    }
}
```

用户手机 service 实现类

```
Java
```

```
package org.qiyu.live.user.provider.service.impl;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import jakarta.annotation.Resource;
import org.qiyu.live.common.interfaces.enums.CommonStatusEum;
import org.qiyu.live.common.interfaces.utils.ConvertBeanUtils;
import org.qiyu.live.user.dto.UserPhoneDTO;
import org.qiyu.live.user.provider.dao.mapper.IUserPhoneMapper;
import org.qiyu.live.user.provider.dao.po.UserPhonePO;
import org.qiyu.live.user.provider.service.IUserPhoneService;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * @Author idea
 * @Date: Created in 19:59 2023/6/10
 * @Description
 */
@Service
public class UserPhoneServiceImpl implements IUserPhoneService {

    @Resource
    private IUserPhoneMapper userPhoneMapper;

    @Override
    public UserPhoneDTO insert(String phone, Long userId) {
        UserPhonePO userPhonePO = new UserPhonePO();
        userPhonePO.setUserId(userId);
        userPhonePO.setPhone(phone);

userPhonePO.setStatus(CommonStatusEum.VALID_STATUS.getCode());
        userPhoneMapper.insert(userPhonePO);
        return ConvertBeanUtils.convert(userPhonePO,
UserPhoneDTO.class);
    }

    @Override
    public List<UserPhoneDTO> queryByUserId(Long userId) {
        //底层会走用户手机号的主键索引
        LambdaQueryWrapper<UserPhonePO> queryWrapper = new
LambdaQueryWrapper<>();
        queryWrapper.eq(UserPhonePO::getUserId, userId);
```

```
        queryWrapper.eq(UserPhonePO::getStatus,
CommonStatusEum.VALID_STATUS.getCode());
        return
ConvertBeanUtils.convertList(userPhoneMapper.selectList(queryWrapp
er), UserPhoneDTO.class);
    }

    @Override
    public UserPhoneDTO queryByPhone(String phone) {
        //底层会走用户 id 的辅助索引
        LambdaQueryWrapper<UserPhonePO> queryWrapper = new
LambdaQueryWrapper<>();
        queryWrapper.eq(UserPhonePO::getPhone, phone);
        queryWrapper.eq(UserPhonePO::getStatus,
CommonStatusEum.VALID_STATUS.getCode());
        queryWrapper.orderByDesc(UserPhonePO::getCreateTime);
        queryWrapper.last("limit 1");
        return
ConvertBeanUtils.convert(userPhoneMapper.selectOne(queryWrapper),U
serPhoneDTO.class);
    }
}
```

用户手机 service 接口

```
Java
package org.qiyu.live.user.provider.service;

import org.qiyu.live.user.dto.UserPhoneDTO;

import java.util.List;

/**
 * @Author idea
 * @Date: Created in 19:59 2023/6/10
 * @Description
 */
public interface IUserPhoneService {

    /**
     * 手机号注册
     *
     * @param phone
     * @param userId
     * @return
```

```
     */
    UserPhoneDTO insert(String phone, Long userId);

    /**
     * 更具用户id查询手机信息
     *
     * @param userId
     * @return
     */
    List<UserPhoneDTO> queryByUserId(Long userId);

    /**
     * 根据手机号查询
     *
     * @param phone
     * @return
     */
    UserPhoneDTO queryByPhone(String phone);
}
```

用户手机 service 实现类

```
Java
package org.qiyu.live.user.provider.service.impl;

import
com.baomidou.mybatisplus.core.conditions.query.LambdaQueryWrapper;
import jakarta.annotation.Resource;
import org.qiyu.live.common.interfaces.enums.CommonStatusEum;
import org.qiyu.live.common.interfaces.utils.ConvertBeanUtils;
import org.qiyu.live.user.dto.UserPhoneDTO;
import org.qiyu.live.user.provider.dao.mapper.IUserPhoneMapper;
import org.qiyu.live.user.provider.dao.po.UserPhonePO;
import org.qiyu.live.user.provider.service.IUserPhoneService;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * @Author idea
 * @Date: Created in 19:59 2023/6/10
 * @Description
 */
@Service
public class UserPhoneServiceImpl implements IUserPhoneService {
```

```
    @Resource
    private IUserPhoneMapper userPhoneMapper;

    @Override
    public UserPhoneDTO insert(String phone, Long userId) {
        UserPhonePO userPhonePO = new UserPhonePO();
        userPhonePO.setUserId(userId);
        userPhonePO.setPhone(phone);

userPhonePO.setStatus(CommonStatusEum.VALID_STATUS.getCode());
        userPhoneMapper.insert(userPhonePO);
        return ConvertBeanUtils.convert(userPhonePO,
UserPhoneDTO.class);
    }

    @Override
    public List<UserPhoneDTO> queryByUserId(Long userId) {
        //底层会走用户手机号的主键索引
        LambdaQueryWrapper<UserPhonePO> queryWrapper = new
LambdaQueryWrapper<>();
        queryWrapper.eq(UserPhonePO::getUserId, userId);
        queryWrapper.eq(UserPhonePO::getStatus,
CommonStatusEum.VALID_STATUS.getCode());
        return
ConvertBeanUtils.convertList(userPhoneMapper.selectList(queryWrapp
er), UserPhoneDTO.class);
    }

    @Override
    public UserPhoneDTO queryByPhone(String phone) {
        //底层会走用户 id 的辅助索引
        LambdaQueryWrapper<UserPhonePO> queryWrapper = new
LambdaQueryWrapper<>();
        queryWrapper.eq(UserPhonePO::getPhone, phone);
        queryWrapper.eq(UserPhonePO::getStatus,
CommonStatusEum.VALID_STATUS.getCode());
        queryWrapper.orderByDesc(UserPhonePO::getCreateTime);
        queryWrapper.last("limit 1");
        return
ConvertBeanUtils.convert(userPhoneMapper.selectOne(queryWrapper),U
serPhoneDTO.class);
    }
```

```
}
```

数据库 PO 对象：

```
Java
package org.qiyu.live.user.provider.dao.po;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;

import java.util.Date;

/**
 * @Author idea
 * @Date: Created in 20:01 2023/6/10
 * @Description
 */
@TableName("t_user_phone")
public class UserPhonePO {

    @TableId(type = IdType.INPUT)
    private String phone;
    private Long userId;
    private Integer status;
    private Date createTime;
    private Date updateTime;

    public Long getUserId() {
        return userId;
    }

    public void setUserId(Long userId) {
        this.userId = userId;
    }

    public String getPhone() {
        return phone;
    }

    public void setPhone(String phone) {
        this.phone = phone;
    }

    public Integer getStatus() {
```

```
        return status;
    }

    public void setStatus(Integer status) {
        this.status = status;
    }

    public Date getCreateTime() {
        return createTime;
    }

    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }

    public Date getUpdateTime() {
        return updateTime;
    }

    public void setUpdateTime(Date updateTime) {
        this.updateTime = updateTime;
    }

    @Override
    public String toString() {
        return "UserPhonePO{" +
                "userId=" + userId +
                ", phone='" + phone + '\'' +
                ", status=" + status +
                ", createTime=" + createTime +
                ", updateTime=" + updateTime +
                '}';
    }
}
```

加解密工具类

```
Java
package org.qiyu.live.common.interfaces.utils;

import java.security.InvalidKeyException;
```

```
import java.security.Key;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;

import org.apache.commons.codec.binary.Base64;

public class DESUtils {

    // 算法名称
    public static final String KEY_ALGORITHM = "DES";
    // 算法名称/加密模式/填充方式
    // DES 共有四种工作模式-->>ECB：电子密码本模式、CBC：加密分组链接模
式、CFB：加密反馈模式、OFB：输出反馈模式
    public static final String CIPHER_ALGORITHM =
"DES/ECB/PKCS5Padding";
    public static final String PUBLIC_KEY = "BAS9j2C3D4E5F60708";

    /**
     * 生成密钥 key 对象
     *
     * @param keyStr 密钥字符串
     * @return 密钥对象
     * @throws InvalidKeyException
     * @throws NoSuchAlgorithmException
     * @throws InvalidKeySpecException
     * @throws Exception
     */
    private static SecretKey keyGenerator(String keyStr) throws
Exception {
        byte input[] = HexString2Bytes(keyStr);
        DESKeySpec desKey = new DESKeySpec(input);
        // 创建一个密匙工厂，然后用它把 DESKeySpec 转换成
        SecretKeyFactory keyFactory =
SecretKeyFactory.getInstance("DES");
        SecretKey securekey = keyFactory.generateSecret(desKey);
        return securekey;
    }
```

```
    private static int parse(char c) {
        if (c >= 'a')
            return (c - 'a' + 10) & 0x0f;
        if (c >= 'A')
            return (c - 'A' + 10) & 0x0f;
        return (c - '0') & 0x0f;
    }

    // 从十六进制字符串到字节数组转换
    public static byte[] HexString2Bytes(String hexstr) {
        byte[] b = new byte[hexstr.length() / 2];
        int j = 0;
        for (int i = 0; i < b.length; i++) {
            char c0 = hexstr.charAt(j++);
            char c1 = hexstr.charAt(j++);
            b[i] = (byte) ((parse(c0) << 4) | parse(c1));
        }
        return b;
    }

    /**
     * 加密数据
     *
     * @param data 待加密数据
     * @return 加密后的数据
     */
    public static String encrypt(String data) throws Exception {
        Key deskey = keyGenerator(PUBLIC_KEY);
        // 实例化 Cipher 对象，它用于完成实际的加密操作
        Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
        SecureRandom random = new SecureRandom();
        // 初始化 Cipher 对象，设置为加密模式
        cipher.init(Cipher.ENCRYPT_MODE, deskey, random);
        byte[] results = cipher.doFinal(data.getBytes());
        // 执行加密操作。加密后的结果通常都会用 Base64 编码进行传输
        return Base64.encodeBase64String(results);
    }

    /**
     * 解密数据
     *
     * @param data 待解密数据
     * @return 解密后的数据
     */
```

```
    public static String decrypt(String data) throws Exception {
        Key deskey = keyGenerator(PUBLIC_KEY);
        Cipher cipher = Cipher.getInstance(CIPHER_ALGORITHM);
        // 初始化 Cipher 对象，设置为解密模式
        cipher.init(Cipher.DECRYPT_MODE, deskey);
        // 执行解密操作
        return new
String(cipher.doFinal(Base64.decodeBase64(data)));
    }

    public static void main(String[] args) throws Exception {
        String phone = "17818723311";
        String encryptStr = DESUtils.encrypt(phone);
        String decryStr = DESUtils.decrypt(encryptStr);
        System.out.println(encryptStr);
        System.out.println(decryStr);
    }

}
```

分表相关的 Nacos 配置：

```
Java
dataSources:
  user_master:  ##新表，重建的分表
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://cloud.db:8808/qiyu_live_user?
useUnicode=true&characterEncoding=utf8
    username: root
    password: root

  user_slave0:  ##新表，重建的分表
    dataSourceClassName: com.zaxxer.hikari.HikariDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
    jdbcUrl: jdbc:mysql://cloud.db:8809/qiyu_live_user?
useUnicode=true&characterEncoding=utf8
    username: root
    password: root

rules:
  - !READWRITE_SPLITTING
    dataSources:
```

```
      user_ds:
        staticStrategy:
          writeDataSourceName: user_master
          readDataSourceNames:
            - user_slave0
  - !SINGLE
    defaultDataSource: user_ds ## 不分表分分库的默认数据源
  - !SHARDING
    tables:
      t_user:
        actualDataNodes: user_ds.t_user_${(0..99).collect()
{it.toString().padLeft(2,'0')}}
        tableStrategy:
          standard:
            shardingColumn: user_id
            shardingAlgorithmName: t_user-inline
      t_user_tag:
        actualDataNodes: user_ds.t_user_tag_${(0..99).collect()
{it.toString().padLeft(2,'0')}}
        tableStrategy:
          standard:
            shardingColumn: user_id
            shardingAlgorithmName: t_user_tag-inline
      t_user_phone:
        actualDataNodes: user_ds.t_user_phone_${(0..99).collect()
{it.toString().padLeft(2,'0')}}
        tableStrategy:
          standard:
            shardingColumn: user_id
            shardingAlgorithmName: t_user_phone-inline
    shardingAlgorithms:
      t_user-inline:
        type: INLINE
        props:
          algorithm-expression: t_user_${(user_id %
100).toString().padLeft(2,'0')}
      t_user_tag-inline:
        type: INLINE
        props:
          algorithm-expression: t_user_tag_${(user_id %
100).toString().padLeft(2,'0')}
      t_user_phone-inline:
        type: INLINE
        props:
```

```
          algorithm-expression: t_user_phone_${(user_id %
100).toString().padLeft(2,'0')}
props:
  sql-show: true
```