6-15 | 登录注册流程完善--短信验证流程

1.实现短信发送接口，当用户输入手机号之后，可以点击短信发送按钮，后台模拟短信发送过程，并且记录到表中。

2.点击登录接口，后台先判断短信是否合法，合法后进行手机号是否存在判断，如果存在，则进行登录，如果不存在就先注册再登录。

3.返回 token 给到前端，前端将 token 保存在本地的 cookie 中

建立消息服务模块

1. qiyu-live-msg-provider
1. qiyu-live-msg-interface

消息服务-短信模块接口设计

1.支持根据手机号发送短信内容

2.支持根据手机号+验证码的方式进行校验

3.支持短信发送记录的存储持久化

基础配置

引入相关依赖，同时创建好 bootstrap.yml,logback-spring.xml,以及 nacos 上相关的配置文件：

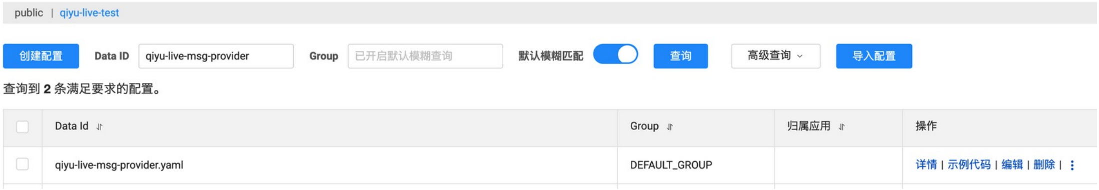
bootstrap.yml

```yaml
YAML
spring:
  cloud:
    nacos:
      username: qiyu
      password: qiyu
      discovery:
        server-addr: qiyu.nacos.com:8848
        namespace: qiyu-live-test
      config:
        import-check:
          enabled: false
        # 当前服务启动后去 nacos 中读取配置文件的后缀
        file-extension: yaml
        # 读取配置的 nacos 地址
        server-addr: qiyu.nacos.com:8848
        # 读取配置的 nacos 的名空间
        namespace: qiyu-live-test
  config:
    import:
      - optional:nacos:qiyu-live-msg-provider.yaml
```

logback-spring.xml

```
XML
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <springProperty name="APP_NAME" scope="context"
source="spring.application.name" defaultValue="undefined"/>
    <!-- 用于生成一个标识，防止多个 Docker 容器映射到同一台宿主机上出现
目录名重复问题-->
    <define name="index"
class="org.qiyu.live.common.interfaces.utils.IpLogConversionRule"/
>
    <property name="LOG_HOME" value="/tmp/logs/${APP_NAME}/$
{index}"/>
    <property name="LOG_PATTERN" value="[%d{yyyy-MM-dd
HH:mm:ss.SSS} -%5p] %-40.40logger{39} :%msg%n"/>

    <!--  控制台标准继续输出内容  -->
    <appender name="CONSOLE"
class="ch.qos.logback.core.ConsoleAppender">
        <!-- 日志输出的格式  -->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>${LOG_PATTERN}</pattern>
        </layout>
    </appender>

    <!--   info 级别的日志，记录到对应的文件内 -->
    <appender name="INFO_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_HOME}/${APP_NAME}.log</file>
        <!-- 滚动策略，日志生成的时候会按照时间来进行分类，例如 2023-
```

```
05-11 日的日志，后缀就会有 2023-05-11，每天的日志归档后的名字都不一样
-->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_HOME}/${APP_NAME}.log.%d{yyyy-
MM-dd}</fileNamePattern>
            <!--  日志只保留 1 个月 -->
            <maxHistory>1</maxHistory>
        </rollingPolicy>
        <!-- 日志输出的格式  -->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>${LOG_PATTERN}</pattern>
        </layout>
    </appender>

    <!--  error 级别的日志，记录到对应的文件内  -->
    <appender name="ERROR_FILE"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${LOG_HOME}/${APP_NAME}_error.log</file>
        <!-- 滚动策略，日志生成的时候会按照时间来进行分类，例如 2023-
05-11 日的日志，后缀就会有 2023-05-11，每天的日志归档后的名字都不一样
-->
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
            <fileNamePattern>${LOG_HOME}/${APP_NAME}_error.log.
%d{yyyy-MM-dd}</fileNamePattern>
            <!--  日志只保留 1 个月 -->
            <maxHistory>1</maxHistory>
        </rollingPolicy>
        <!-- 日志输出的格式  -->
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>${LOG_PATTERN}</pattern>
        </layout>
        <!--   值记录 error 级别的日志 -->
        <filter class="ch.qos.logback.classic.filter.LevelFilter">
            <level>error</level>
            <onMismatch>DENY</onMismatch>
        </filter>
    </appender>

    <!-- 根输出级别为 INFO，控制台中将出现包含 info 及以上级别的日志-->
    <!-- 日志输出级别 -->
    <root level="INFO">
        <!-- ref 值与上面的 appender 标签的 name 相对应 -->
        <appender-ref ref="CONSOLE"/>
        <appender-ref ref="INFO_FILE"/>
        <appender-ref ref="ERROR_FILE"/>
    </root>
</configuration>
```



查询到 2 条满足要求的配置。

□	Data Id	Group	归属应用	操作
□	qiyu-live-msg-provider.yaml	DEFAULT_GROUP		详情 | 示例代码 | 编辑 | 删除 | ⋮

nacos 上的配置文件：

```
SQL
spring:
  application:
    name: qiyu-live-msg-provider
  datasource:
    hikari:
      minimum-idle: 10
      maximum-pool-size: 200
    driver-class-name: com.mysql.cj.jdbc.Driver
    #访问主库
    url: jdbc:mysql://数据库地址:3306/qiyu_live_msg?
useUnicode=true&characterEncoding=utf8
    username: root
    password: root
  data:
    redis:
      port: 8801
      host: cloud.db
      lettuce:
        pool:
          min-idle: 10
          max-active: 100
          max-idle: 10
dubbo:
  application:
    name: ${spring.application.name}
  registry:
    address: nacos://nacos 地址:8848?namespace=qiyu-live-
test&&username=qiyu&&password=qiyu
```

```
  protocol:
    name: dubbo
    port: 9092
```

maven 依赖

SQL

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.idea</groupId>
        <artifactId>qiyu-live-app</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <artifactId>qiyu-live-msg-provider</artifactId>
    <description>消息服务</description>

    <properties>
        <sharding.jdbc.version>5.3.2</sharding.jdbc.version>
        <mybatis-plus.version>3.5.3</mybatis-plus.version>
        <qiyu-live-msg-interface.version>1.0-SNAPSHOT</qiyu-live-
msg-interface.version>
        <qiyu-live-redis-starter.version>1.0-SNAPSHOT</qiyu-live-
redis-starter.version>
        <qiyu-live-common-interface.version>1.0-SNAPSHOT</qiyu-
live-common-interface.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.idea</groupId>
            <artifactId>qiyu-live-msg-interface</artifactId>
            <version>${qiyu-live-msg-interface.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.dubbo</groupId>
            <artifactId>dubbo-spring-boot-starter</artifactId>
            <version>${dubbo.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
config</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-bootstrap</artifactId>
            <version>${spring-cloud-boostrap.version}</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-starter-alibaba-nacos-
config</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
            <exclusions>
                <exclusion>
                    <artifactId>log4j-to-slf4j</artifactId>
                    <groupId>org.apache.logging.log4j</groupId>
                </exclusion>
            </exclusions>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${qiyu-mysql.version}</version>
        </dependency>
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>${mybatis-plus.version}</version>
        </dependency>
        <dependency>
            <groupId>org.idea</groupId>

<artifactId>qiyu-live-framework-redis-starter</artifactId>
            <version>${qiyu-live-redis-starter.version}</version>
        </dependency>
        <dependency>
            <groupId>org.idea</groupId>
            <artifactId>qiyu-live-common-interface</artifactId>

<version>${qiyu-live-common-interface.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shardingsphere</groupId>
            <artifactId>shardingsphere-jdbc-core</artifactId>
            <version>${sharding.jdbc.version}</version>
        </dependency>

    </dependencies>
</project>
```

代码内容

启动类

```
Java
package org.qiyu.live.msg.provider;

import
org.apache.dubbo.config.spring.context.annotation.EnableDubbo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.WebApplicationType;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import
org.springframework.cloud.client.discovery.EnableDiscoveryClient;

/**
 * @Author idea
 * @Date: Created in 17:21 2023/6/11
 * @Description
 */
@SpringBootApplication
@EnableDiscoveryClient
@EnableDubbo
public class MsgProviderApplication {

    public static void main(String[] args) {
        SpringApplication springApplication = new
SpringApplication(MsgProviderApplication.class);

springApplication.setWebApplicationType(WebApplicationType.NONE);
        springApplication.run(args);
    }
}
```

短信服务的相关 rpc 接口

```
Java
package org.qiyu.live.msg.provider.interfaces;

import org.qiyu.live.msg.provider.dto.MsgCheckDTO;
import org.qiyu.live.msg.provider.enums.MsgSendResultEnum;

/**
 * @Author idea
 * @Date: Created in 17:21 2023/6/11
 * @Description
 */
public interface ISmsRpc {

    /**
     * 发送短信接口
     *
     * @param phone
     * @return
     */
    MsgSendResultEnum sendMessage(String phone);

    /**
     * 校验登录验证码
     *
     * @param phone
     * @param code
     * @return
     */
    MsgCheckDTO checkLoginCode(String phone, Integer code);

    /**
     * 插入一条短信记录
     *
     * @param phone
     * @param code
     */
    void insertOne(String phone, Integer code);
}
```

rpc 实现类

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

import jakarta.annotation.Resource;
import org.apache.dubbo.config.annotation.DubboService;
import org.qiyu.live.msg.provider.dto.MsgCheckDTO;
import org.qiyu.live.msg.provider.enums.MsgSendResultEnum;
import org.qiyu.live.msg.provider.interfaces.ISmsRpc;
import org.qiyu.live.msg.provider.service.ISmsService;

/**
 * @Author idea
 * @Date: Created in 17:20 2023/6/11
 * @Description
 */
@DubboService
public class SmsRpcImpl implements ISmsRpc {

    @Resource
    private ISmsService smsService;

    @Override
    public MsgSendResultEnum sendMessage(String phone) {
        return smsService.sendMessage(phone);
    }

    @Override
    public MsgCheckDTO checkLoginCode(String phone, Integer code)
{
        return smsService.checkLoginCode(phone,code);
    }

    @Override
    public void insertOne(String phone, Integer code) {
        smsService.insertOne(phone,code);
    }
}
```

service 层接口代码

Java

```
package org.qiyu.live.msg.provider.service;

import org.qiyu.live.msg.provider.dto.MsgCheckDTO;
import org.qiyu.live.msg.provider.enums.MsgSendResultEnum;

/**
 * @Author idea
 * @Date: Created in 17:30 2023/6/11
 * @Description
 */
public interface ISmsService {

    /**
     * 发送短信接口
     *
     * @param phone
     * @return
     */
    MsgSendResultEnum sendMessage(String phone);

    /**
     * 校验登录验证码
     *
     * @param phone
     * @param code
     * @return
     */
    MsgCheckDTO checkLoginCode(String phone, Integer code);

    /**
     * 插入一条短信记录
     *
     * @param phone
     * @param code
     */
    void insertOne(String phone, Integer code);
}
```

service 层实现类代码

Java

```java
package org.qiyu.live.msg.provider.service.impl;

import jakarta.annotation.Resource;
import org.apache.commons.lang3.RandomUtils;
import org.apache.commons.lang3.StringUtils;
import org.idea.qiyu.live.framework.redis.starter.key.MsgProviderCacheKeyBuilder;
import org.qiyu.live.msg.provider.config.ThreadPoolManager;
import org.qiyu.live.msg.provider.dao.mapper.SmsMapper;
import org.qiyu.live.msg.provider.dao.po.SmsPO;
import org.qiyu.live.msg.provider.dto.MsgCheckDTO;
import org.qiyu.live.msg.provider.enums.MsgSendResultEnum;
import org.qiyu.live.msg.provider.service.ISmsService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.data.redis.core.RedisTemplate;
import org.springframework.stereotype.Service;

import java.util.concurrent.TimeUnit;

/**
 * @Author idea
 * @Date: Created in 17:30 2023/6/11
 * @Description
 */
@Service
public class SmsServiceImpl implements ISmsService {

    private static Logger logger = LoggerFactory.getLogger(SmsServiceImpl.class);

    @Resource
    private SmsMapper smsMapper;
    @Resource
    private RedisTemplate<String, Integer> redisTemplate;
    @Resource
    private MsgProviderCacheKeyBuilder redisKeyBuilder;

    @Override
    public MsgSendResultEnum sendMessage(String phone) {
        //模拟产生一条短信记录
        int code = RandomUtils.nextInt(100000, 999999);
        String key = redisKeyBuilder.buildSmsLoginCodeKey(phone);
        boolean hasKey = redisTemplate.hasKey(key);
        if (hasKey) {
            logger.error("短期内同一个手机号不能发送太多次，phone is
{}", phone);
            //短期内同一个手机号不能发送太多次
            return MsgSendResultEnum.SEND_FAIL;
        }
        redisTemplate.opsForValue().set(key, code);
        redisTemplate.expire(key, 60, TimeUnit.SECONDS);
        ThreadPoolManager.commonAsyncPool.execute(() -> {
            //模拟发送短信信息
            mockSendSms(phone, code);
            insertOne(phone, code);
        });
        return MsgSendResultEnum.SEND_SUCCESS;
    }

    @Override
    public MsgCheckDTO checkLoginCode(String phone, Integer code)
{
        String key = redisKeyBuilder.buildSmsLoginCodeKey(phone);
        Integer recordCode = redisTemplate.opsForValue().get(key);
        if (recordCode == null) {
            return new MsgCheckDTO(false, "验证码已失效");
        }
        boolean isCodeVerify = recordCode.equals(code);
        if (isCodeVerify) {
            redisTemplate.delete(key);
            return new MsgCheckDTO(true, "");
        }
        return new MsgCheckDTO(false, "验证码校验失败");
    }

    @Override
    public void insertOne(String phone, Integer code) {
        SmsPO smsPO = new SmsPO();
        smsPO.setPhone(phone);
        smsPO.setCode(code);
        smsMapper.insert(smsPO);
    }

    /**
     * 模拟发送短信过程，感兴趣的朋友可以尝试对接一些第三方的短信平台
     *
     * @param phone
     * @param code
     */
    private void mockSendSms(String phone, Integer code) {
        try {
            logger.info(" ============= 创建短信发送通道中 =========
==== ,phone is {},code is {}", phone, code);
            Thread.sleep(1000);
            logger.info(" ============= 短信已经发送成功 ==========
=== ");
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

}
```

dao 层的 mapper 接口

```
Java
package org.qiyu.live.msg.provider.dao.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import org.apache.ibatis.annotations.Mapper;
import org.qiyu.live.msg.provider.dao.po.SmsPO;

/**
 * @Author idea
 * @Date: Created in 17:26 2023/6/11
 * @Description
 */
@Mapper
public interface SmsMapper extends BaseMapper<SmsPO> {
}
```

dao 层的 po 对象

```
Java
package org.qiyu.live.msg.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;

/**
 * 记录短信相关信息
 *
 * @Author idea
 * @Date: Created in 17:26 2023/6/11
 * @Description
 */
@TableName("t_sms")
public class SmsPO {

    @TableId(type = IdType.AUTO)
    private Long id;
    private Integer code;
    private String phone;
    private Date sendTime;
    private Date updateTime;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public Integer getCode() {
        return code;
    }

    public void setCode(Integer code) {
        this.code = code;
    }

    public String getPhone() {
        return phone;
    }

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

    public Date getSendTime() {
        return sendTime;
    }

    public void setSendTime(Date sendTime) {
        this.sendTime = sendTime;
    }

    public Date getUpdateTime() {
        return updateTime;
    }

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

    @Override
    public String toString() {
        return "SmsPO{" +
                "id=" + id +
                ", code='" + code + '\'' +
                ", phone='" + phone + '\'' +
                ", sendTime=" + sendTime +
                ", updateTime=" + updateTime +
                '}';
    }
}
```

配置类

建立了一个统一的线程池管理工具类

```
Java
package org.qiyu.live.msg.provider.config;

import java.util.concurrent.*;

/**
 * @Author idea
 * @Date: Created in 17:38 2023/6/11
 * @Description
 */
public class ThreadPoolManager {

    public static ThreadPoolExecutor commonAsyncPool = new
ThreadPoolExecutor(2, 8, 3, TimeUnit.SECONDS, new
ArrayBlockingQueue<>(1000)
            , new ThreadFactory() {
        @Override
        public Thread newThread(Runnable r) {
            Thread newThread = new Thread(r);
            newThread.setName(" commonAsyncPool - " +
ThreadLocalRandom.current().nextInt(10000));
            return newThread;
        }
    });

}
```

sql

短信记录表建表 sql

```
SQL
CREATE TABLE `t_sms` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT COMMENT '主键id',
  `code` int unsigned DEFAULT '0' COMMENT '验证码',
  `phone` varchar(200) CHARACTER SET utf8mb4 COLLATE
utf8mb4_0900_ai_ci DEFAULT '' COMMENT '手机号',
  `sendTime` datetime DEFAULT CURRENT_TIMESTAMP COMMENT '发送时间',
  `updateTime` datetime DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP COMMENT '更新时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci;
```

interface 层的一些补充

MsgCheckDTO 对象

```
SQL
package org.qiyu.live.msg.provider.dto;

import java.io.Serial;
import java.io.Serializable;

/**
 * @Author idea
 * @Date: Created in 07:41 2023/6/12
 * @Description
 */
public class MsgCheckDTO implements Serializable {

    @Serial
    private static final long serialVersionUID =
3394248744287019717L;
    private boolean checkStatus;
    private String desc;


    public MsgCheckDTO(boolean checkStatus, String desc) {
        this.checkStatus = checkStatus;
        this.desc = desc;
    }

    public boolean isCheckStatus() {
        return checkStatus;
    }

    public void setCheckStatus(boolean checkStatus) {
        this.checkStatus = checkStatus;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

    @Override
    public String toString() {
        return "MsgCheckDTO{" +
                "checkStatus=" + checkStatus +
                ", desc='" + desc + '\'' +
                '}';
    }
}
```

短信发送结果枚举

```
SQL
```

```
package org.qiyu.live.msg.provider.enums;

/**
 * @Author idea
 * @Date created in 7:28 下午 2022/11/23
 */
public enum MsgSendResultEnum {

    SEND_SUCCESS(0,"成功"),
    SEND_FAIL(1,"发送失败"),
    MSG_PARAM_ERROR(2,"消息格式异常");

    int code;
    String desc;

    MsgSendResultEnum(int code, String desc) {
        this.code = code;
        this.desc = desc;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getDesc() {
        return desc;
    }

    public void setDesc(String desc) {
        this.desc = desc;
    }

}
```