消息发送中心

学习目标:

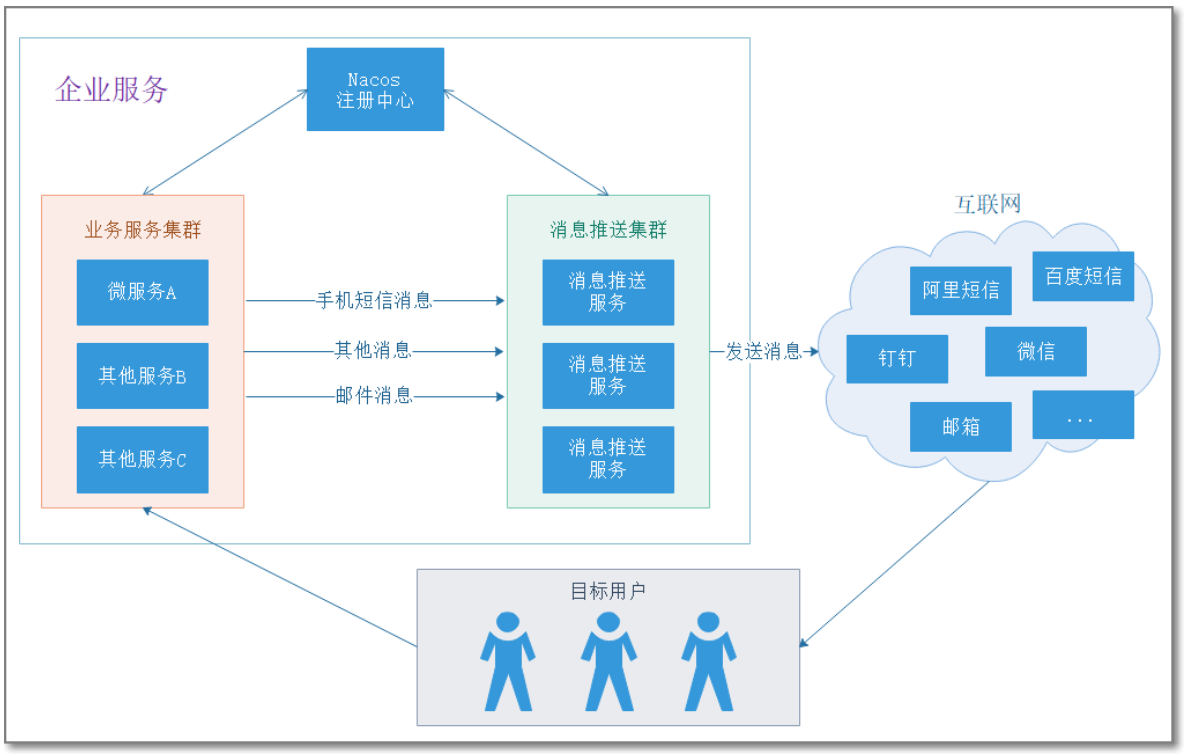
- 了解消息发送的需求
- 了解JavaMail
- 使用HuTool发送邮件消息
- 使用阿里邮件发送消息
- 实现单个邮件发送
- 实现批量邮件发送
- 实现同步邮件发送
- 实现异步邮件发送
- 实现邮件API接口
- 实现短信发送
- 实现短信API接口

1. 项目介绍

1.1 需求分析

在进行企业级开发的时候，我们往往需要发送各种消息给用户，例如手机短信、邮件、微信、钉钉消息等。

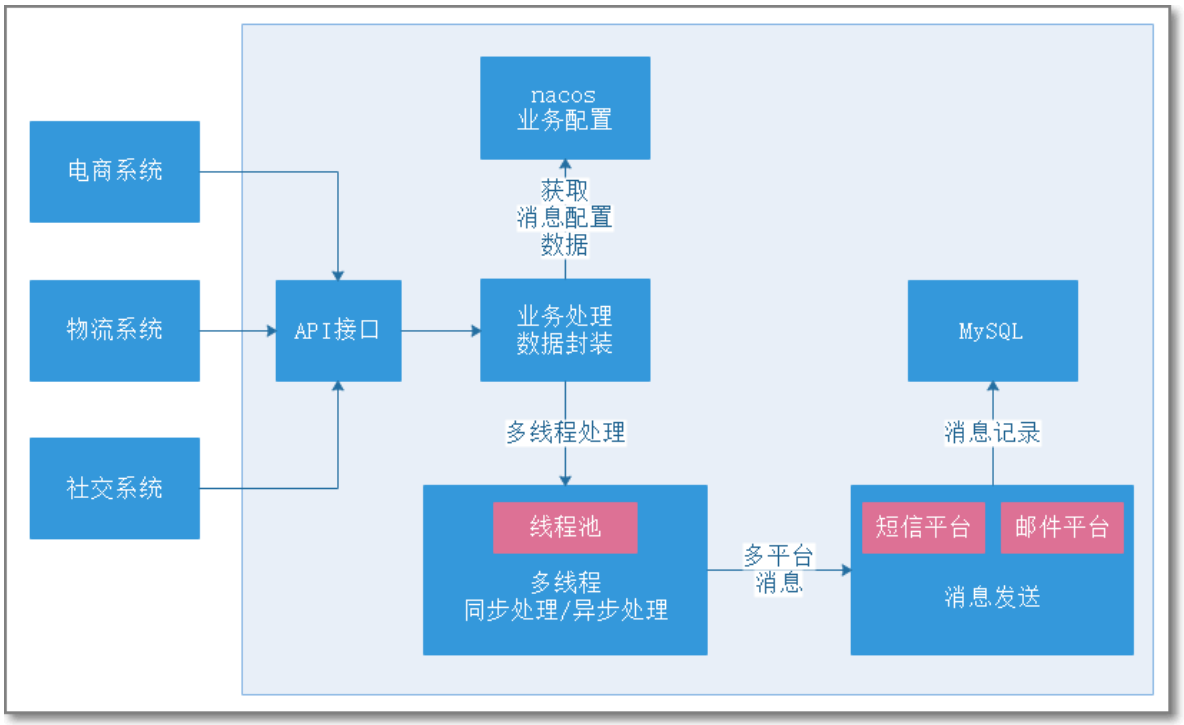
而每次都进行重复开发，是一件很麻烦且繁琐的事情。我们希望能有一个中间服务，负责处理所有类似的消息发送，而其他的服务不用关系消息发送的细节，只需要调用接口就能够实现消息的发送。由专门的团队进行消息发送的研发以及升级维护。



1.2 系统架构

为了提高性能，需要使用多线程进行消息发送。根据实际情况，分为同步处理和异步处理。

把消息发送抽取为一个模块，每个实现对应一种消息平台。本项目虽然只实现邮件和短信消息，但是可以很方便的进行扩充，以便支持其他的消息服务。



1.3 数据库表

邮件消息和手机短信消息分别对应两张数据库表：

邮件消息表：

Name	Comment	Data Type
id	主键ID	varchar(20)
app_name	应用名称	varchar(20)
msg_type	消息类型	tinyint
msg_name	消息接收者	varchar(50)
template	模板名称	varchar(20)
subject	邮件标题	varchar(50)
content	邮件内容	text
success	发送状态	tinyint
result_msg	发送结果	varchar(200)
create_time	创建时间	datetime
modified_time	修改时间	datetime

短信消息表：

Code	Comment	Data Type
id	主键ID	varchar(20)
app_name	应用名称	varchar(20)
msg_type	消息类型	tinyint
msg_phone	消息接收电话	text
template_id	消息模板ID	varchar(20)
content	消息内容	varchar(200)
success	发送状态	tinyint
result_msg	发送结果	varchar(200)

Code	Comment	Data Type
create_time	创建时间	datetime
modified_time	修改时间	datetime

1.4 开发环境准备

1. JDK版本：1.8及以上
2. Maven：3.X以上
3. 开发工具：IDEA
4. 服务器：CentOS 7.5
5. Docker：1.13
6. 数据库：MySQL 8.0.18

准备MySQL：

```
docker run -id --name msg_mysql \
-v ~/mysql/data:/var/lib/mysql \
-p 3306:3306 \
-e MYSQL_ROOT_PASSWORD=root -d mysql:8.0.18
```

执行资料中的sql文件，导入数据库表

准备nacos：

Nacos（Dynamic *Na*ming and *C*onfiguration *S*ervice） 是阿里巴巴推出的用于发现、配置和管理微服务。Nacos 提供了一组简单易用的特性集，帮助您快速实现动态**服务发现、服务配置**、服务元数据及流量管理。

启动Nacos：

```
#启动容器
docker run --env MODE=standalone --name msg_nacos \
-d -p 8848:8848 nacos/nacos-server:1.1.4
```

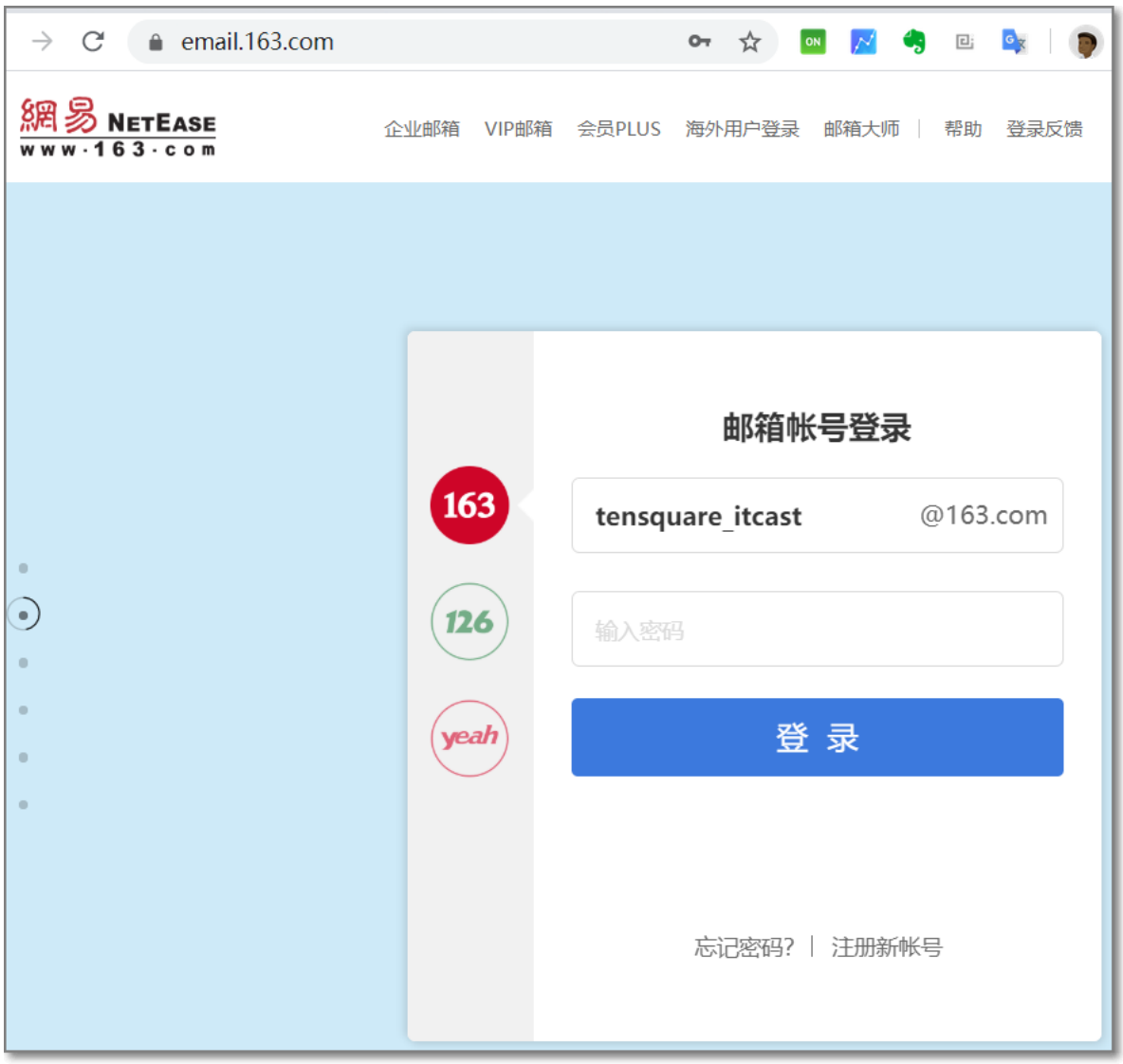
访问地址：http://{虚拟机IP}:8848/nacos
账户和密码都是 nacos

2. 发送邮件入门

2.1 电子邮件介绍

2.1.1 简介

电子邮件是一种用电子手段提供信息交换的通信方式，是互联网应用最广的服务。电子邮件可以是文字、图像、声音等多种形式。同时，用户可以得到大量免费的新闻、专题邮件，并轻松实现轻松的信息搜索。电子邮件的存在极大地方便了人与人之间的沟通与交流，促进了社会的发展。



现在很多服务商都提供了邮件服务，我们在这里主要介绍如何使用程序进行邮件的发送。

2.1.2 JavaMail介绍

JDK本身就提供了发送邮件的功能，就是JavaMail。

JavaMail API提供了独立于平台和协议的框架来构建邮件和消息传递应用程序。JavaMail API作为与Java SE平台一起使用的可选软件包提供，并且也包含在Java EE平台中。

官网：https://www.oracle.com/technetwork/java/javamail/index.html

不过现在已经移动到了github进行维护：

github网址：https://javaee.github.io/javamail/

2.1.3 电子邮件协议

但是要使用JavaMail发邮件，前提是先要有一个邮件服务，利用电子邮件协议进行邮件的发送。我们需要先简单了解邮件协议：

SMTP 简单邮件传输
SMTP是Simple Mail Transfer Protocol的简称，即**简单邮件传输协议**。该协议定义了邮件客户端软件和SMTP服务器之间，以及两台SMTP服务器之间的通信规则。在Java Mail API中，基于Java Mail的程序将与本公司或Internet服务提供商（ISP）的SMTP服务器通信，该SMTP服务器将会把消息转发给用于接收消息的SMTP服务器，最后用户可通过POP或IMAP协议获取该消息。由于**支持身份验证**，所以不需要SMTP服务器是一种开放的转发器，但需要确保SMTP服务器配置正确。Java Mail API中没有集成用于处理配置服务器以转发消息或添加/删除电子邮件账户等功能。

POP3邮局协议
POP3是Post Office Protocol 3的简称，即邮局协议第3版，该协议是在RFC 1939中定义的。它是Internet电子邮件的第一个离线协议标准，也就是说，POP3是一个简单而实用的邮件信息传输协议。目前在Internet上，大多数人都采用POP3来接收邮件。**POP3为每个用户的每个邮箱提供支持**，但是POP3并不支持查看邮件信息及统计新邮件的功能，因此，在使用Java Mail API时，如果想获取这些信息，需要自己进行计算。

IMAP 接收邮件协议
IMA P的含义是Internet消息访问协议，当前版本是第4版，也称作IMAP4，该协议是在RFC 2060中定义的。IMAP是**接收消息的更加高级协议**。使用IMAP时，所用邮件服务器必须支持该协议，但是如果所有邮件服务器只支持该协议，并不能使用全部IMAP协议，为了能够支持全部IMAP协议，所用邮件服务器**还应该支持POP协议**。在Java Mail程序通过IMAP协议可以在服务器上拥有包括多个文件夹的用户，并且这些文件夹可以被多个用户共享。
虽然IMAP协议具有更高级的功能，但是也不要让每个用户都使用，这是因为IMAP需要服务器接收新消息，发送消息给请求的用户，并在多个文件夹中为每个用户维护消息、备份信息，这会加重邮件服务器的负荷。

MIME
MIME是多用途的网际邮件扩充协议的简称。它不是一种邮件传输协议，而是用于定义传输内容（如消息的格式、附件等）的协议。许多文档都对MIME协议作了描述（包括RFC 2045、RFC 2046和RFC 2047）。MIME是对RFC 822的扩充，RFC 822规定了内容只包括采用ASCII编码的纯文本的邮件格式，而MIME允许在邮件中包含附件。作为Java Mail API的用户，一般不需要担心这些格式。

2.2 HuTool

虽然JavaMail是一种非常普遍的发送邮件的API，但是应用起来非常的麻烦。

我们在这里使用对JavaMail进行封装的工具HuTool。

2.2.2 简介

Hutool是一个小而全的Java工具类库，通过静态方法封装，降低相关API的学习成本，提高工作效率，使Java拥有函数式语言般的优雅，让Java语言也可以“甜甜的”。

Hutool中的工具方法来自于每个用户的精雕细琢，它涵盖了Java开发底层代码中的方方面面，它既是大型项目开发中解决小问题的利器，也是小型项目中的效率担当；

Hutool是项目中“util”包友好的替代，它节省了开发人员对项目中公用类和公用工具方法的封装时间，使开发专注于业务，同时可以最大限度的避免封装不完善带来的bug。

Hutool = Hu + tool，是原公司项目底层代码剥离后的开源库，“Hu”是公司名称的表示，tool表示工具。Hutool谐音“糊涂”，一方面简洁易懂，一方面寓意“难得糊涂”。

官网：https://www.hutool.cn/

文档网址：https://www.hutool.cn/docs/

API网址：https://apidoc.gitee.com/loolly/hutool/

2.2.2 申请163邮箱

前面说过，要使用API发送邮件，首先要有一个邮箱服务，我们使用163的邮箱服务。

申请163邮箱：(`https://mail.163.com/`)

打开邮箱中心：

开启邮箱服务：

注意要获取授权密码，并记录密码，后面要使用密码。

2.2.3 实现邮件发送

创建父工程msg_send，在其中创建子模块demo进行测试，在demo的pom.xml中添加依赖：

```
<!--Hutoolg工具类库-->
<dependency>
    <groupId>cn.hutool</groupId>
    <artifactId>hutool-all</artifactId>
    <version>5.2.5</version>
</dependency>
<!--JavaMail-->
<dependency>
    <groupId>javax.mail</groupId>
    <artifactId>mail</artifactId>
    <version>1.4.7</version>
</dependency>
```

实现编码：

```
public static void main(String[] args) {
    MailAccount account = new MailAccount();
    account.setHost("smtp.163.com");
    account.setPort(25);
    account.setAuth(true);
    account.setFrom("tensquare_itcast@163.com");
    account.setUser("tensquare_itcast@163.com");
    account.setPass("IDGPFWRTPMRFWLXB"); //密码

    String send = MailUtil.send(account,
"tensquare_itcast@163.com",
                                "测试", "邮件来自Hutool测试",
false);

    System.out.println(send);
}
```

效果如下：

特点：

- 需要自己准备邮件服务器
- 邮件发送灵活，没有限制
- 开发成本高，很多功能需要自己实现
- 发送邮件不需要额外费用

2.3 阿里邮件推送

2.3.1 简介

阿里的邮件推送（DirectMail）是一款简单高效的电子邮件发送服务，它构建在可靠稳定的阿里云基础之上，帮助您快速、精准地实现事务邮件、通知邮件和批量邮件的发送。

邮件发送历经两年双11考验，在发送速度、系统稳定性和到达率上表现优异；提供丰富的接口和灵活的使用方式，为企业和开发者解决邮件投递的难题，用户无需自建邮件服务器。

- **稳定和可扩展性**

基于云的电子邮件技术，分布式部署
外发邮件冗余存储到多台服务器和数据库

- **快速和精准性**

  高并发处理邮件
  策略化的邮件发送功能，有效控制发信频率
  在最快的投递速度下保证高质量的到达

- **多种邮件类型**

  触发邮件：注册确认、订单确认、密码重置、交易通知等
  批量邮件：自有用户的产品促销、期刊订阅等

- **高性价比**

  按邮件发送量付费，没有最低消费

2.3.2 开通邮件发送

申请地址：https://www.aliyun.com/product/directmail

需要提前申请域名。使用数据跟踪功能，需先进行域名备案，发信域名只有验证通过状态，才可使用该功能。

1. **配置发件域名**

发信配置

请至域名 mail.xuziyi.top 服务商处添加以下三项域名解析配置，配置后生效大概需要1~10分钟不等，请点击"刷新"按钮查看配置结果

*1、所有权验证

类型	主机记录	主域名	记录值	状态
TXT	aliyundm.mail	xuziyi.top	8317e21a2e004ffab1c9	验证通过

*2、spf验证

类型	主机记录	主域名	记录值	状态
TXT	mail	xuziyi.top	v=spf1 include:spf1.dm.aliyun.com -all	验证通过

说明：如果您已经添加了SPF验证，则请在SPF的TXT记录中加上 include:spf1.dm.aliyun.com。

收信配置

请至域名 mail.xuziyi.top DNS服务提供商处添加MX记录，并保持，否则会无法发信。

*3、MX验证

类型	主机记录	主域名	记录值	状态
MX	mail	xuziyi.top	mx01.dm.aliyun.com	验证通过

跟踪配置

请至域名mail.xuziyi.top DNS服务提供商处添加CNAME记录。

4、CNAME验证

类型	主机记录	主域名	记录值	状态
CNAME	dmtrace .mail	xuziyi.top	tracedm.aliyuncs.com	验证通过

2. 配置发件地址

发信地址

发信地址 ▾ 发信地址 搜索

类型	发信地址	状态	回信地址（状态）
批量邮件	msg@mail.xuziyi.top	正常	tensquare_itcast@163.com(通过)
触发邮件	admin@mail.xuziyi.top	正常	tensquare_itcast@163.com(通过)

说明：
1. 发信地址最多可以创建10个。
2. 触发类邮件指注册激活、密码找回等；批量邮件指营销推广、订阅期刊等。不同类型邮件的发送限制不同，请根据邮件类型选择。
3. 回信地址验证通过后才生效，否则系统将分配一个随机的地址，您无法收到用户回复。
4. 设置SMTP密码在2分钟内生效。
5. SMTP服务地址： smtpdm.aliyun.com ， SMTP服务端口号：25或80或465(SSL加密)。

2.3.3 获取AccessKey

在使用阿里云提供的短信、邮箱等服务的时候，需要申请AccessKey，才可以正常使用阿里云的服务

申请地址：https://usercenter.console.aliyun.com/#/manage/ak

其中AccessKey ID和Access Key Secret就是我们后续要使用的key

```
AccessKey:  LTAI4FpSmghKjkTkPeS55cwc
Access Key Secret:  Y6OWohY13Bdh8uBoRukecQus1BhNOf
```

2.3.3 使用SDK发送

参考网址：https://help.aliyun.com/document_detail/29459.html

添加依赖：

```
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>4.5.0</version>
    <scope>compile</scope>
</dependency>
<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-dm</artifactId>
    <version>3.3.1</version>
    <scope>compile</scope>
</dependency>
```

实现功能：

```
public static void main(String[] args) {
    //客户端
    IClientProfile profile = DefaultProfile.getProfile(
            "cn-hangzhou", "LTAI4FpSmghKjkTkPeS55cwc",
"Y6OWohY13Bdh8uBoRukecQus1BhNOf");
    IAcsClient client = new DefaultAcsClient(profile);
    SingleSendMailRequest request = new SingleSendMailRequest();
    try {
        request.setAccountName("admin@mail.xuziyi.top");
        request.setFromAlias("管理员");

        request.setAddressType(1);
        request.setReplyToAddress(true);
        request.setToAddress("tensquare_itcast@163.com");
        request.setSubject("邮件发送");
        request.setHtmlBody("<h3>阿里邮件发送测试正文</h3>");
        request.setMethod(MethodType.POST);

        client.getAcsResponse(request);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

效果如下：

3. 创建工程

3.1 创建父工程

创建maven父工程msg_send，在pom.xml中添加如下依赖：

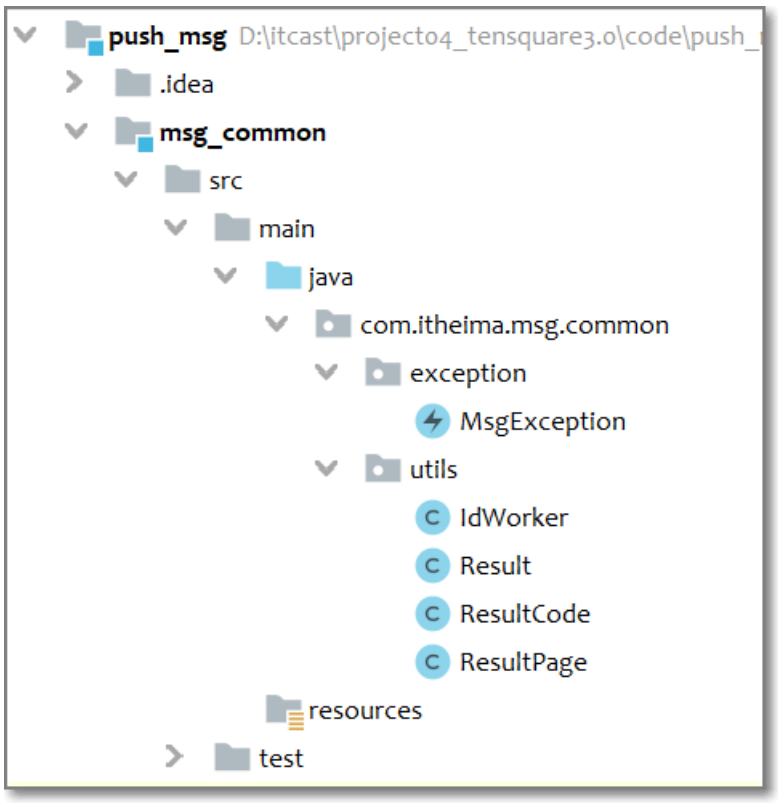
```
<packaging>pom</packaging>

<!-- Spring boot 版本信息 -->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.6.RELEASE</version>
    <relativePath/>
</parent>

<properties>
    <!-- Environment Settings -->
    <java.version>1.8</java.version>
    <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
    <!-- Spring Cloud 版本信息 -->
    <spring-cloud.version>Greenwich.SR2</spring-cloud.version>
    <!--  Spring Cloud Alibaba 版本信息 -->

 <spring.cloud.alibaba.version>2.1.0.RELEASE</spring.cloud.alibab
a.version>
    <swagger2.version>2.9.2</swagger2.version>

    <lombok.version>1.18.10</lombok.version>
    <mybatis-plus-boot-starter.version>3.3.0</mybatis-plus-boot-
starter.version>
    <hutool-all.version>5.2.5</hutool-all.version>
    <mail.version>1.4.7</mail.version>
    <aliyun-java-sdk-core.version>4.5.0</aliyun-java-sdk-
core.version>
    <aliyun-java-sdk-dm.version>3.3.1</aliyun-java-sdk-
dm.version>
</properties>
```

```
<dependencyManagement>
    <dependencies>
        <!-- Spring Cloud begin-->
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!-- Spring Cloud end-->
        <!-- Spring Cloud Alibaba begin-->
        <dependency>
            <groupId>com.alibaba.cloud</groupId>
            <artifactId>spring-cloud-alibaba-
dependencies</artifactId>
            <version>${spring.cloud.alibaba.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
        <!-- Spring Cloud Alibaba end -->
    </dependencies>
</dependencyManagement>

<dependencies>
    <!-- nacos注册中心依赖包 -->
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-
discovery</artifactId>
    </dependency>
    <!-- 监控检查-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <!-- nacos配置中心依赖支持 -->
    <dependency>
        <groupId>com.alibaba.cloud</groupId>
        <artifactId>spring-cloud-starter-alibaba-nacos-
config</artifactId>
    </dependency>

    <dependency>
```

```
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <version>${lombok.version}</version>
        <scope>provided</scope>
    </dependency>


    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger2</artifactId>
        <version>${swagger2.version}</version>
    </dependency>

    <dependency>
        <groupId>io.springfox</groupId>
        <artifactId>springfox-swagger-ui</artifactId>
        <version>${swagger2.version}</version>
    </dependency>

</dependencies>

<!-- 阿里云镜像 -->
<repositories>
    <repository>
        <id>aliyun</id>
        <name>aliyun</name>

 <url>http://maven.aliyun.com/nexus/content/groups/public</url>
        <releases>
            <enabled>true</enabled>
        </releases>
        <snapshots>
            <enabled>true</enabled>
        </snapshots>
    </repository>
</repositories>
```

3.2 导入公共工程

创建msg_common子模块，并在资源中找到common工程的资源放到项目中：



执行maven命令：

mvn install -Dmaven.test.skip=true

3.3 创建消息发送工程

3.3.1 添加依赖

创建maven子模块msg_web，在pom.xml中添加依赖：

```
<dependencies>
    <dependency>
        <groupId>com.itheima</groupId>
        <artifactId>msg_common</artifactId>
        <version>1.0-SNAPSHOT</version>
        <scope>compile</scope>
    </dependency>

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

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

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

    <!-- mybatis-plus begin-->
    <dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus-boot-starter</artifactId>
        <version>${mybatis-plus-boot-starter.version}</version>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>

    <!-- Hutoolg工具 -->
    <dependency>
        <groupId>cn.hutool</groupId>
        <artifactId>hutool-all</artifactId>
        <version>${hutool-all.version}</version>
    </dependency>
    <!-- JavaMail -->
    <dependency>
        <groupId>javax.mail</groupId>
        <artifactId>mail</artifactId>
        <version>${mail.version}</version>
    </dependency>

    <!-- 阿里SDK -->
    <dependency>
        <groupId>com.aliyun</groupId>
        <artifactId>aliyun-java-sdk-core</artifactId>
        <version>${aliyun-java-sdk-core.version}</version>
    </dependency>
    <dependency>
        <groupId>com.aliyun</groupId>
```

```
        <artifactId>aliyun-java-sdk-dm</artifactId>
        <version>${aliyun-java-sdk-dm.version}</version>
    </dependency>

</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

3.3.2 编写基础类

在com.itheima.msg.web中编写引导类：

```
@EnableSwagger2
@SpringBootApplication
@EnableDiscoveryClient
public class MsgApplication {
    public static void main(String[] args) {
        SpringApplication.run(MsgApplication.class, args);
    }
}
```

在com.itheima.msg.web.config中添加MyBatisPlus配置类：

```
@Configuration
public class MybatisPlusConfig {
    @Bean
    public PaginationInterceptor paginationInterceptor() {
        return new PaginationInterceptor();
    }
}
```

```
/**
 * 自定义ID生成器
 */
```

```
@Component
public class CustomIdGenerator implements IdentifierGenerator {
    @Bean
    public IdWorker idWorkker(){
        return new IdWorker(1, 1);
    }

    @Autowired
    private IdWorker idWorker;

    @Override
    public Long nextId(Object entity) {
        return idWorker.nextId();
    }
}
```

```
/**
 * 地址：http://localhost:8801/swagger-ui.html
 */
@Configuration
@EnableSwagger2
public class SwaggerConfig extends WebMvcConfigurationSupport {

    //是否开启swagger，正式环境一般是需要关闭的，可根据springboot的多环境配置进行设置
    @Value(value = "${swagger.enabled}")
    Boolean swaggerEnabled;

    @Bean
    public Docket createRestApi() {
        // 文档类型
        return new Docket(DocumentationType.SWAGGER_2)
                .groupName("消息发送中心")
                // 创建api的基本信息
                .apiInfo(apiInfo())
                // 选择哪些接口去暴露
                .select()
                // 扫描的包

.apis(RequestHandlerSelectors.basePackage("com.itheima.msg.web.controller"))
                .paths(PathSelectors.any())
                .build()
                .enable(swaggerEnabled);
    }
```

```
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("消息发送中心--Swagger2文档")
                .version("1.0")
                .build();
    }

    /**
     * 防止@EnableMvc把默认的静态资源路径覆盖了，手动设置的方式
     *
     * @param registry
     */
    @Override
    protected void addResourceHandlers(ResourceHandlerRegistry
registry) {
        // 解决静态资源无法访问

 registry.addResourceHandler("/**").addResourceLocations("classpa
th:/static/");
        // 解决swagger无法访问
        registry.addResourceHandler("/swagger-
ui.html").addResourceLocations("classpath:/META-INF/resources/");
        // 解决swagger的js文件无法访问

 registry.addResourceHandler("/webjars/**").addResourceLocations(
"classpath:/META-INF/resources/webjars/");

    }

}
```

在com.itheima.msg.web.util中编写MsgType：

```
/**
 * <pre>
 * 消息类型常量
 * </pre>
 */
public enum MsgType {
    /**
     * 消息类型
     */
```

```
    MP_TEMPLATE(1, "公众号-模板消息"),
    MA_TEMPLATE(2, "小程序-模板消息"),
    KEFU(3, "公众号-客服消息"),
    KEFU_PRIORITY(4, "公众号-客服消息优先"),
    ALI_YUN(5, "阿里云短信"),
    TX_YUN(7, "腾讯云短信"),
    YUN_PIAN(8, "云片网短信"),
    UP_YUN(9, "又拍云短信"),
    HW_YUN(10, "华为云短信"),
    EMAIL(11, "E-Mail"),
    WX_CP(12, "微信企业号/企业微信"),
    HTTP(13, "HTTP请求"),
    DING(14, "钉钉"),
    BD_YUN(15, "百度云短信"),
    QI_NIU_YUN(16, "七牛云短信"),
    WX_UNIFORM_MESSAGE(17, "小程序-统一服务消息"),
    MA_SUBSCRIBE(18, "小程序-订阅消息");

    private int code;
    private String name;

    public int getCode() {
        return code;
    }

    public String getName() {
        return name;
    }

    MsgType(int code, String name) {
        this.code = code;
        this.name = name;
    }
}
```

3.3.2 配置文件

在auth子模块中添加配置文件：

在resources目录中创建配置文件bootstrap.yml内容为：

```yaml
spring:
  application:
    name: msg-send
  profiles:
    active: dev
```

创建配置文件bootstrap-dev.yml内容为：

```yaml
spring:
  cloud:
    # 使用 Naoos 作为服务注册发现
    nacos:
      discovery:
        server-addr: 192.168.200.188:8848
      config:
        server-addr: 192.168.200.188:8848
        group: MSG_GROUP
        file-extension: yml
server:
  port: 8801
```

创建配置文件logback-spring.xml内容为：

```xml
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <include
resource="org/springframework/boot/logging/logback/defaults.xml"
/>
    <include
resource="org/springframework/boot/logging/logback/console-
appender.xml" />
    <property name="LOG_FILE" value="logs/cold-user.log" /> <!--
log目录 -->
    <appender name="ROLLING"
class="ch.qos.logback.core.rolling.RollingFileAppender">
        <encoder>
            <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS} %-5level
[%thread] %logger{36}[%L] - %msg%n </pattern> <!-- log打印的格式 --
>
        </encoder>
        <file>${LOG_FILE}</file>
        <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
```

```
 <fileNamePattern>${LOG_FILE}.%d{yyyyMMdd}.%i</fileNamePattern>
<!-- log文件名格式 -->
            <maxHistory>30</maxHistory> <!-- 磁盘保留log文件最大个数
-->
            <timeBasedFileNamingAndTriggeringPolicy
class="ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP">
                <maxFileSize>512MB</maxFileSize> <!-- 单个log文件的
最大文件大小 -->
            </timeBasedFileNamingAndTriggeringPolicy>
        </rollingPolicy>
    </appender>
    <!--定义控制台日志级别-->
    <logger name="com.itheima" level="DEBUG" additivity="false">
        <appender-ref ref="CONSOLE"/>
    </logger>
    <root level="INFO">
        <appender-ref ref="CONSOLE" />
        <appender-ref ref="ROLLING" />
    </root>
</configuration>
```

在nacos中新建鉴权服务配置：

Data ID：

```
msg-send-dev.yml
```

Group：

```
MSG_GROUP
```

配置格式：

```
YAML
```

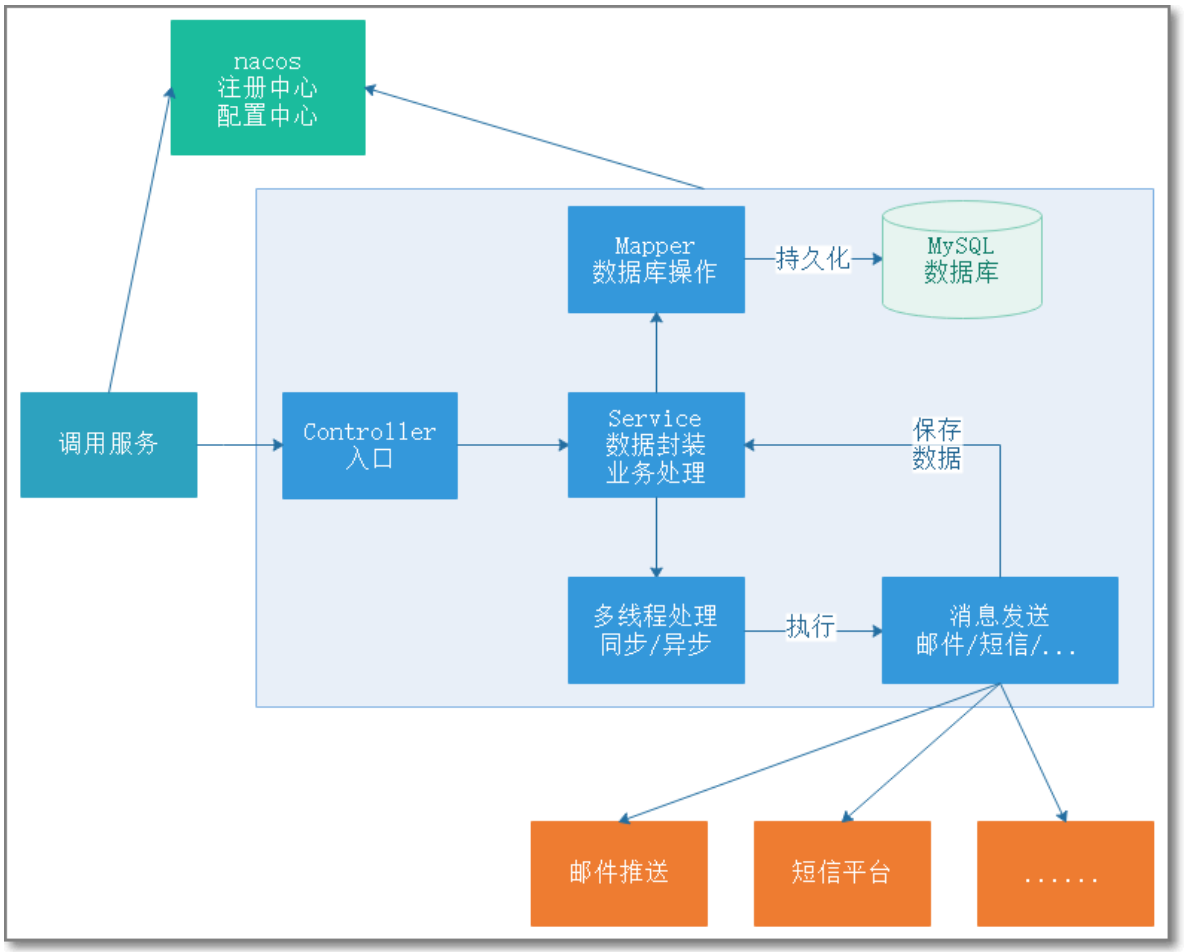
配置内容：

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    password: root
```

```
    url: jdbc:mysql://192.168.200.188:3306/msg?
serverTimezone=Asia/Shanghai
    username: root

# 是否开启swagger
swagger:
  enabled: true

mybatis-plus:
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

4. 实现邮件发送

在开发之前，我们先要弄清楚我们到底要做什么，下图中浅蓝部分的区域就是我们主要开发的部分：



我们开发从最后端开始，就是消息发送部分，以便于大家的理解。

4.1 实现邮件发送

4.1.1 添加配置

在nacos的配置中添加：

```
aliyun:
  accessKey: LTAI4FpSmghKjkTkPeS55cwc
  accessKeySecret: Y6OWohY13Bdh8uBoRukecQus1BhNOf
  maxRequestsPerHost: 20 #每个Host最大连接数
```

4.1.2 编写DTO

在com.itheima.msg.web.dto中编写dto：

```
@Data
@Builder
@AllArgsConstructor(access = AccessLevel.PRIVATE)
@NoArgsConstructor
public class AliMailMsgDto {

    //应用名称
    private String appName;

    //------单个邮件-----
    //触发邮件的发件地址
    private String triggerName;
    //触发邮件的发件昵称
    private String triggerAlias;
    //目标邮箱地址
    private String address;
    //邮件主题
    private String subject;
    //邮件正文
    private String content;
}
```

4.1.3 实现消息发送

在com.itheima.msg.web.sender中编写MsgSender接口：

```
/**
 * <pre>
 * 消息发送器接口
 * </pre>
 */
public interface MsgSender<T> {

    /**
     * 发送消息
     *
     * @param msgData 消息数据
     */
    Result send(T msgData);

}
```

在com.itheima.msg.web.logic.sender中编写AliMailMsgSender：

```
/**
 * <pre>
 * E-Mail发送器
 * </pre>
 */
@Slf4j
@Component
public class AliMailMsgSender implements MsgSender<AliMailMsgDto>
{

    @Value("${aliyun.accessKey}")
    private String accessKey;
    @Value("${aliyun.accessKeySecret}")
    private String accessSecret;
    @Value("${aliyun.maxRequestsPerHost}")
    private int maxRequestsPerHost;

    /**
     * 阿里云短信client
     */
    private volatile static IAcsClient iAcsClient;

    @Override
```

```
    public Result send(AliMailMsgDto msgData) {
        Result result = null;
        try {
            SingleSendMailRequest request = new
SingleSendMailRequest();
            request.setAccountName(msgData.getTriggerName());
            request.setFromAlias(msgData.getTriggerAlias());
            request.setAddressType(1);
            request.setReplyToAddress(true);
            request.setToAddress(msgData.getAddress());
            request.setSubject(msgData.getSubject());
            request.setHtmlBody(msgData.getContent());
            request.setMethod(MethodType.POST);
            getClient().getAcsResponse(request);

            result = result.builder()
                    .status(ResultCode.OK)
                    .msg("邮件发送成功")
                    .build();

        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
            result = result.builder()
                    .status(ResultCode.ERROR)
                    .msg("邮件发送异常")
                    .data(ExceptionUtils.getMessage(e))
                    .build();
        }
        return result;
    }

    /**
     * 获取阿里云邮件发送客户端
     *
     * @return IAcsClient
     */
    private IAcsClient getClient() {
        if (iAcsClient == null) {
            synchronized (AliMailMsgSender.class) {
                if (iAcsClient == null) {
                    // 创建DefaultAcsClient实例并初始化
                    DefaultProfile profile =
DefaultProfile.getProfile(
                            "cn-hangzhou", accessKey, accessSecret);
```

```
                    // 每个host的最大连接数，超时时间等
                    HttpClientConfig clientConfig =
HttpClientConfig.getDefault();

 clientConfig.setMaxRequestsPerHost(maxRequestsPerHost);

 clientConfig.setConnectionTimeoutMillis(10000L);

                    profile.setHttpClientConfig(clientConfig);
                    iAcsClient = new DefaultAcsClient(profile);
                }
            }
        }
        return iAcsClient;
    }
}
```

4.2 保存发送信息

4.2.1 编写持久层

邮件消息数据库表：

Name	Comment	Data Type
id	主键ID	varchar(20)
app_name	应用名称	varchar(20)
msg_type	消息类型	tinyint
msg_name	消息接收者	varchar(50)
template	模板名称	varchar(20)
subject	邮件标题	varchar(50)
content	邮件内容	text
success	发送状态	tinyint
result_msg	发送结果	varchar(200)
create_time	创建时间	datetime
modified_time	修改时间	datetime

在com.itheima.msg.web.entity中编写BaseEntity：

```
@Getter
@Setter
public abstract class BaseEntity implements Serializable {

    private String appName;

    private Integer msgType;

    private Date createTime;

    private Date modifiedTime;
}
```

在com.itheima.msg.web.entity中编写MailMsg：

```
@Getter
@Setter
@TableName("t_mail_msg")
public class MailMsg extends BaseEntity {
    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "主键ID")
    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private String id;

    private String msgName;

    private String template;

    private String subject;

    private String content;

    private boolean success;

    private String resultMsg;
}
```

在com.itheima.msg.web.mapper中编写MailMsgMapper：

```
@Mapper
public interface MailMsgMapper extends BaseMapper<MailMsg> {
}
```

4.2.2 编写Service业务层

在com.itheima.msg.web.service中编写MailMsgService：

```
public interface MailMsgService extends IService<MailMsg> {
    void saveResult(AliMailMsgDto aliMailMsgDto, Result result);
}
```

在com.itheima.msg.web.service.impl中编写MailMsgServiceImpl：

```
@Slf4j
@Service
public class MailMsgServiceImpl extends
ServiceImpl<MailMsgMapper, MailMsg> implements MailMsgService {

    @Override
    @Transactional
    public void saveResult(AliMailMsgDto msgDto, Result result) {
        MailMsg mailMsg = new MailMsg();
        mailMsg.setAppName(msgDto.getAppName());
        mailMsg.setMsgType(MsgType.EMAIL.getCode());

        mailMsg.setMsgName(msgDto.getAddress());
        mailMsg.setSubject(msgDto.getSubject());
        mailMsg.setContent(msgDto.getContent());

        mailMsg.setSuccess(result.getStatus() == ResultCode.OK);
        mailMsg.setResultMsg(result.getData() != null ?
result.getData().toString() : "");

        save(mailMsg);
    }
}
```

4.2.3 改造消息发送

改造AliMailMsgSender，添加保存数据：

```
@Autowired
private MailMsgService mailMsgService;

/**
     * 阿里云短信client
     */
private volatile static IAcsClient iAcsClient;

@Override
public Result send(AliMailMsgDto msgData) {
    Result result = null;
    try {
        SingleSendMailRequest request = new
SingleSendMailRequest();
        request.setAccountName(msgData.getTriggerName());
        request.setFromAlias(msgData.getTriggerAlias());
        request.setAddressType(1);
        request.setReplyToAddress(true);
        request.setToAddress(msgData.getAddress());
        request.setSubject(msgData.getSubject());
        request.setHtmlBody(msgData.getContent());
        request.setMethod(MethodType.POST);
        getClient().getAcsResponse(request);

        result = result.builder()
            .status(ResultCode.OK)
            .msg("邮件发送成功")
            .build();

    } catch (Exception e) {
        log.error(ExceptionUtils.getStackTrace(e));
        result = result.builder()
            .status(ResultCode.ERROR)
            .msg("邮件发送异常")
            .data(ExceptionUtils.getMessage(e))
            .build();
    }

    try {
        //保存数据
        mailMsgService.saveResult(msgData, result);
    } catch (Exception e) {
        log.error(ExceptionUtils.getStackTrace(e));
```

```
    }

    return result;
}
```

4.2.4 自动处理时间

修改BaseEntity：

```
@TableField(fill = FieldFill.INSERT)
private Date createTime;

@TableField(fill = FieldFill.INSERT_UPDATE)
private Date modifiedTime;
```

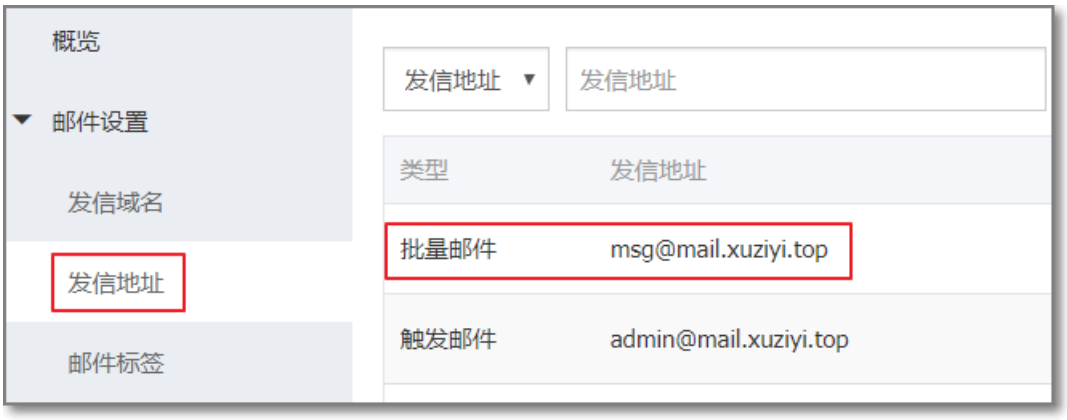
在com.itheima.msg.web.config中添加TimeMetaObjectHandler

```
/**
 * 自动填充处理类
 **/
@Component
public class TimeMetaObjectHandler implements MetaObjectHandler {


    @Override
    public void insertFill(MetaObject metaObject) {
        this.setFieldValByName("createTime",new Date(),
metaObject);
        this.setFieldValByName("modifiedTime", new Date(),
metaObject);
    }

    @Override
    public void updateFill(MetaObject metaObject) {
        this.setFieldValByName("modifiedTime", new Date(),
metaObject);
    }
}
```

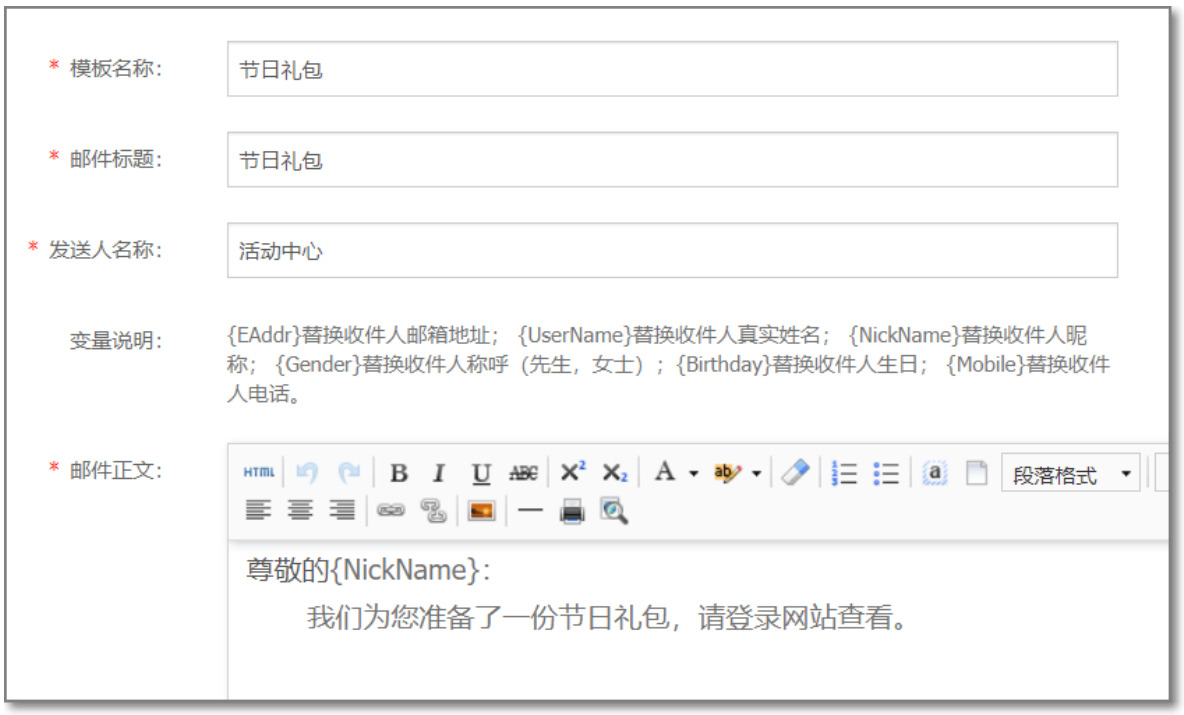
4.3 批量发送邮件

4.3.1 申请批量发送资源

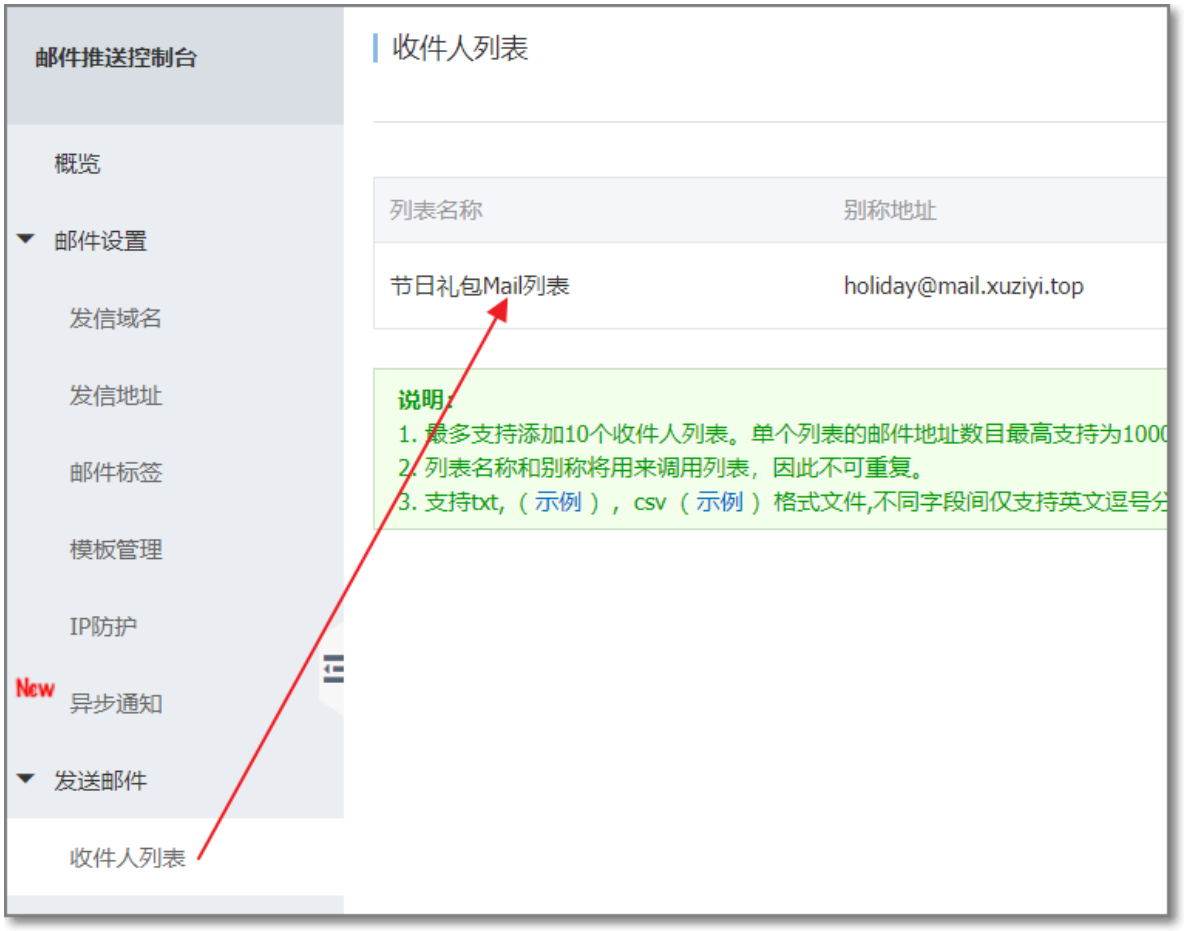
在阿里云邮件控制台申请发信地址：



申请模板：



申请收件人列表:



4.3.2 实现批量发送功能

测试邮件批量发送功能:

```
private static void sendBatch() {
    IClientProfile profile = DefaultProfile.getProfile("cn-
hangzhou",
            "LTAI4FpSmghKjkTkPeS55cwc",
"Y6OWohY13Bdh8uBoRukecQus1BhNOf");
    IAcsClient client = new DefaultAcsClient(profile);

    try {
        BatchSendMailRequest request = new
BatchSendMailRequest();

        //设置发件人的邮件地址
        request.setAccountName("msg@mail.xuziyi.top");
        //设置模板名称
        request.setTemplateName("节日礼包");
        //设置收件人列表名称
        request.setReceiversName("节日礼包Mail列表");

        request.setAddressType(1);
        request.setMethod(MethodType.POST);

        client.getAcsResponse(request);
    } catch (Exception e) {

    }
}
```

实现批量发送功能：

改造AliMailMsgDto添加以下属性：

```
//------批量消息--------
//批量邮件的发件地址
private String batchName;
//收件人列表名称
private String receivers;
//模板名称
private String template;
```

MsgSender接口添加批量发送方法：

```
/**
 * 批量发送消息
 *
 * @param msgData 消息数据
 */
Result bathSend(T msgData);
```

AliMailMsgSender实现批量发送方法：

```
@Override
public Result bathSend(AliMailMsgDto msgData) {
    Result result = null;
    try {
        BatchSendMailRequest request = new
BatchSendMailRequest();
        request.setAccountName(msgData.getBatchName());
        request.setTemplateName(msgData.getTemplate());
        request.setReceiversName(msgData.getReceivers());
        request.setAddressType(1);
        request.setMethod(MethodType.POST);
        getClient().getAcsResponse(request);

        result = result.builder()
            .status(ResultCode.OK)
            .msg("邮件批量发送成功")
            .build();

    } catch (Exception e) {
        log.error(ExceptionUtils.getStackTrace(e));
        result = result.builder()
            .status(ResultCode.ERROR)
            .msg("邮件批量发送异常")
            .data(ExceptionUtils.getMessage(e))
            .build();
    }

    try {
        //保存数据
        mailMsgService.saveResult(msgData, result);
    } catch (Exception e) {
        log.error(ExceptionUtils.getStackTrace(e));
    }
```

```
    return result;
}
```

MailMsgServiceImpl改造保存方法：

```
public void saveResult(AliMailMsgDto msgDto, Result result) {
    MailMsg mailMsg = new MailMsg();
    mailMsg.setAppName(msgDto.getAppName());
    mailMsg.setMsgType(MsgType.EMAIL.getCode());

    if (StringUtils.isNotBlank(msgDto.getAddress())) {
        //------单个邮件-----
        mailMsg.setMsgName(msgDto.getAddress());
        mailMsg.setSubject(msgDto.getSubject());
        mailMsg.setContent(msgDto.getContent());
    } else {
        //------批量邮件------
        mailMsg.setMsgName(msgDto.getBathchName());
        mailMsg.setTemplate(msgDto.getTemplate());
    }

    mailMsg.setSuccess(result.getStatus() == ResultCode.OK);
    mailMsg.setResultMsg(result.getData() != null ?
result.getData().toString() : "");

    save(mailMsg);
}
```

5. 实现多线程调用

5.1 创建线程池

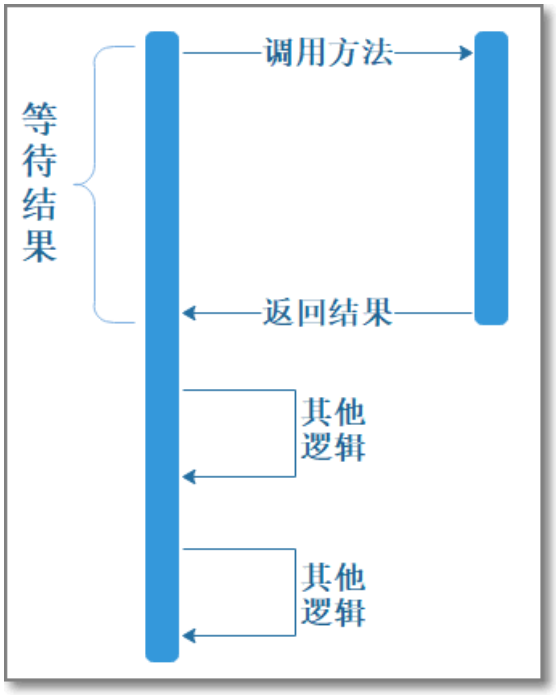
线程是一个程序员一定会涉及到的概念，但是线程的创建和切换都是代价比较大的。所以，我们需要有一个好的方案能做到线程的复用，这就涉及到一个概念——线程池。合理的使用线程池能够带来3个很明显的好处：

1. 降低资源消耗：通过重用已经创建的线程来降低线程创建和销毁的消耗
2. 提高响应速度：任务到达时不需要等待线程创建就可以立即执行。
3. 提高线程的可管理性：线程池可以统一管理、分配、调优和监控。

5.2 实现同步调用

5.2.1 同步调用介绍

普通模式如下图：



测试两种同步方法：

```
public class ThreadSyncDemo {

    //创建线程池
    private static ExecutorService pool =
Executors.newFixedThreadPool(4);

    public static void main(String[] args) {
        //test1();
        test2();

        pool.shutdown();
    }

    private static void test2() {
        System.out.println("test2-1");
        CountDownLatch countDownLatch = new CountDownLatch(2);
```

```
        //要执行的任务1（发送一种消息）
        AtomicInteger b1 = new AtomicInteger(2);
        pool.execute(() -> {
            try {
                b1.addAndGet(2);
                TimeUnit.MILLISECONDS.sleep(20);
                System.out.println("test2-21");
                countDownLatch.countDown();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        //要执行的任务2（发送另一种消息）
        AtomicInteger b2 = new AtomicInteger(20);
        pool.execute(() -> {
            try {
                b2.addAndGet(20);
                TimeUnit.MILLISECONDS.sleep(10);
                System.out.println("test2-22");
                countDownLatch.countDown();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        //不知道任务要执行多久，所以延时的方式不可取
        //try {
        //    TimeUnit.MILLISECONDS.sleep(10);
        //} catch (InterruptedException e) {
        //    e.printStackTrace();
        //}
        try {
            countDownLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println(b1.get() + ":::" + b2.get());
    }

    private static void test1() {
        System.out.println("test1-1");
        //要执行的任务1（发送一种消息）
```

```
        Future<Integer> f1 = pool.submit(() -> {
            int a = 1 + 1;
            TimeUnit.MILLISECONDS.sleep(20);
            System.out.println("test1-21");
            return a;
        });

        //要执行的任务2（发送另一种消息）
        Future<Integer> f2 = pool.submit(() -> {
            int a = 10 + 10;
            TimeUnit.MILLISECONDS.sleep(10);
            System.out.println("test1-22");
            return a;
        });


        try {
            //获取返回结果
            Integer r1 = f1.get();
            Integer r2 = f2.get();
            System.out.println("test1-3");
            System.out.println(r1 + ":::" + r2);
        } catch (InterruptedException e) {
            e.printStackTrace();
        } catch (ExecutionException e) {
            e.printStackTrace();
        }
    }
}
```

使用pool.submit(...)进行同步发送消息，相对于pool.execute(...)有如下好处：

1. 更方便获取执行结果
2. 外部能够很容易的获取执行的异常

5.2.2 实现功能

在com.itheima.msg.web.thread编写MsgThread接口：

```
public interface MsgThread<T> {

    /**
```

```
     * 发送消息
     *
     * @param msgData 消息数据
     */
    Result send(T msgData);

    /**
     * 批量发送消息
     *
     * @param msgData 消息数据
     */
    Result sendBath(T msgData);
}
```

在com.itheima.msg.web.thread编写线程基础类：

```
/**
 * <pre>
 * 消息发送服务线程父类
 * </pre>
 */
@Slf4j
public abstract class BaseMsgThread {

    private static ExecutorService pool;

    static {
        //获取CPU数量
        int cpuCount =
Runtime.getRuntime().availableProcessors();
        //创建固定线程数的线程池
        pool = Executors.newFixedThreadPool(cpuCount * 2);
    }

    public ExecutorService getPool() {
        return pool;
    }
}
```

在com.itheima.msg.web.thread编写AliMailMsgThread：

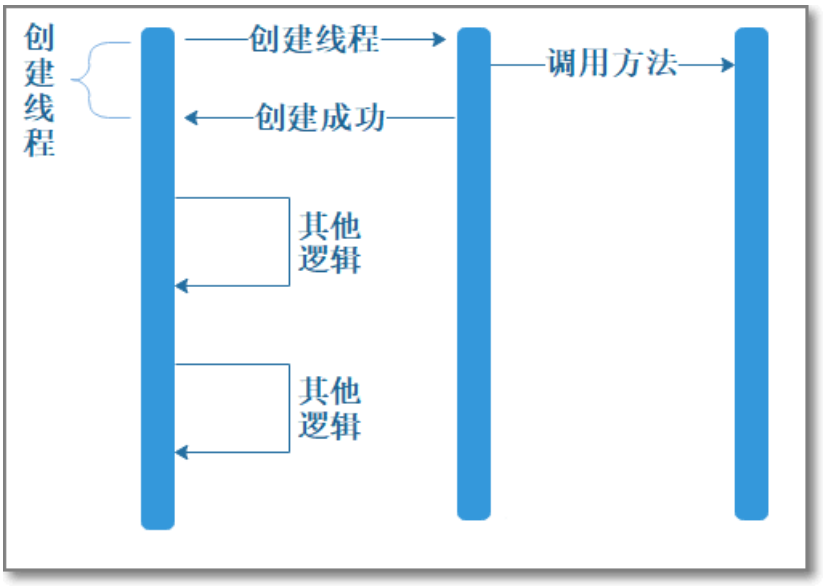
```
/**
```

```
 * <pre>
 * 消息发送服务线程
 * </pre>
 */
@Slf4j
@Component
public class AliMailMsgThread extends BaseMsgThread implements
MsgThread<AliMailMsgDto> {

    @Qualifier("aliMailMsgSender")//可以方便更换其他实现方式
    @Autowired
    private MsgSender msgSender;

    @Override
    public Result send(AliMailMsgDto msgData) {
        Future<Result> submit = getPool().submit(() ->
msgSender.send(msgData));
        try {
            return submit.get();
        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
        }

        return Result.error("服务内部错误");
    }

    @Override
    public Result sendBath(AliMailMsgDto msgData) {
        Future<Result> submit = getPool().submit(() ->
msgSender.bathSend(msgData));
        try {
            return submit.get();
        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
        }

        return Result.error("服务内部错误");
    }
```

5.3 实现异步调用

5.3.1 异步调用介绍

异步模式如下图：



测试两种异步方法：

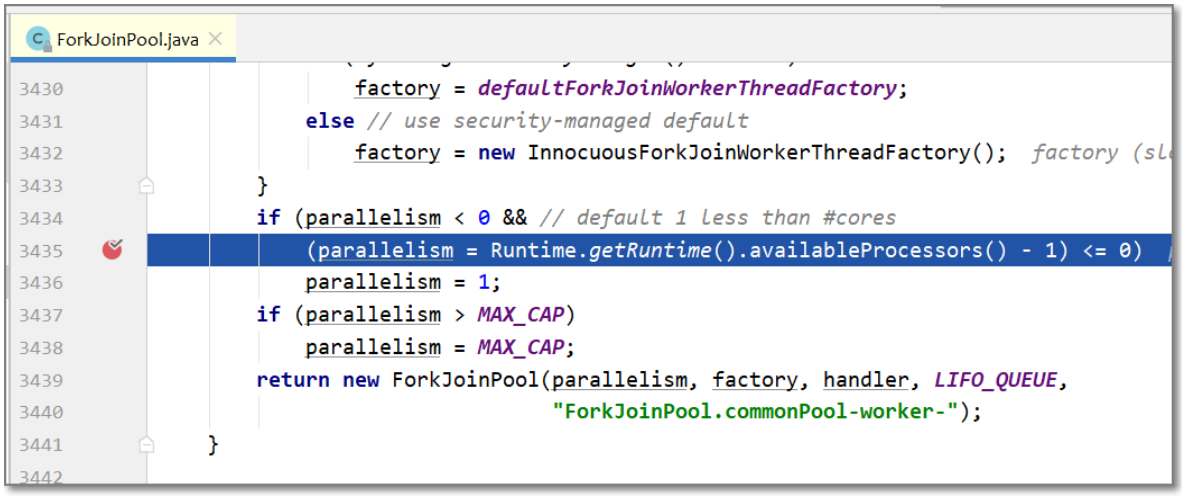
```
public class ThreadAsyncDemo {

    private static ExecutorService pool =
Executors.newFixedThreadPool(4);

    public static void main(String[] args) {
        //test1();

        test2();

        System.out.println("主线程执行完了！");
        pool.shutdown();

        try {
            TimeUnit.MILLISECONDS.sleep(250);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }

    private static void test2() {
        System.out.println("test2开始执行了");
        CompletableFuture.runAsync(() -> {
            try {
                int a = 2 + 2;
                TimeUnit.MILLISECONDS.sleep(100);
                System.out.println("任务1执行的结果" + a);
            } catch (InterruptedException e) {
```

```
                e.printStackTrace();
            }
        });


        CompletableFuture.runAsync(() -> {
            try {
                int a = 20 + 20;
                TimeUnit.MILLISECONDS.sleep(200);
                System.out.println("任务2执行的结果" + a);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        System.out.println("test2执行结束了");
    }

    private static void test1() {
        System.out.println("test1开始执行");
        //任务1
        pool.execute(() -> {
            try {
                int a = 1 + 1;
                TimeUnit.MILLISECONDS.sleep(100);
                System.out.println("任务1的结果：" + a);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });

        //任务2
        pool.execute(() -> {
            try {
                int a = 10 + 10;
                TimeUnit.MILLISECONDS.sleep(200);
                System.out.println("任务2的结果：" + a);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        });


        System.out.println("test1执行完了");
    }
```

```
}
```

`ExecutorService`的线程池可以设置线程数，在线程阻塞的时候，线程池会调度线程池队列中的其他未线程运行

`ForkJoinPool`线程数是当前可用处理器数量-1，其工作机制是不停执行本地任务队列中的任务，任务是一个个取的，顺序执行，没有塞回去的动作，并不会因为某个任务引起阻塞后而换个任务继续执行，所以不适合有阻塞的任务



我们这里是调用三方平台的接口，需要阻塞以等待对方的响应，所以**不适合使用`ForkJoinPool`**。

5.3.2 实现异步调用

在MsgThread中添加异步调用接口方法：

```
    /**
     * 异步发送消息
     *
     * @param msgData 消息数据
     */
    Result asyncSend(T msgData);

    /**
     * 异步批量发送消息
     *
     * @param msgData 消息数据
     */
    Result asyncSendBath(T msgData);
```

在AliMailMsgThread中实现接口方法：

```
@Override
public Result asyncSend(AliMailMsgDto msgData) {
    getPool().execute(() -> msgSender.send(msgData));
    return Result.ok("已经执行成功异步发送消息", null);
}

@Override
public Result asyncSendBatch(AliMailMsgDto msgData) {
    getPool().execute(() -> msgSender.batchSend(msgData));
    return Result.ok("已经执行成功异步批量发送消息", null);
}
```

6. 实现邮件API接口

6.2 接口介绍

我们要完成3个接口

POST　/mail/code/{sync}/{appName}　　发送邮箱验证码

POST　/mail/notify/{sync}/{appName}　　发送邮箱通知

POST　/mail/holiday/{sync}/{appName} 发送节日礼包提醒

请求路径第一个参数使用 sync--同步处理　async--异步处理

请求路径第二个参数appName，传递不同应用的应用名，以方便管理

6.2 实现Controller

在com.itheima.msg.web.controller中复制此异常处理类：

```
import com.itheima.msg.common.exception.MsgException;
import com.itheima.msg.common.utils.Result;
import com.itheima.msg.common.utils.ResultCode;
import lombok.extern.slf4j.Slf4j;
import org.springframework.validation.BindException;
```

```
import org.springframework.validation.ObjectError;
import
org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.validation.ConstraintViolation;
import javax.validation.ConstraintViolationException;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.util.List;
import java.util.Set;

@Slf4j
@ControllerAdvice
public class BaseExceptionHandler {

    @ExceptionHandler(Exception.class)
    @ResponseBody
    public Result handler(Exception e) {
        StringWriter sw = new StringWriter();
        PrintWriter pw = new PrintWriter(sw);
        e.printStackTrace(pw);
        log.error(sw.toString());
        return Result.error(ResultCode.ERROR, "服务错误，请稍候再
试");
    }

    @ExceptionHandler(MsgException.class)
    @ResponseBody
    public Result handler(MsgException e) {
        StringWriter sw = new StringWriter();
        PrintWriter pw = new PrintWriter(sw);
        e.printStackTrace(pw);
        log.error(sw.toString());
        return Result.error(ResultCode.BAD_REQUEST,
e.getMessage());
    }

    @ExceptionHandler(ConstraintViolationException.class)
    @ResponseBody
    public Result handler(ConstraintViolationException e) {
        StringBuffer errorMsg = new StringBuffer();
```

```
        Set<ConstraintViolation<?>> violations =
e.getConstraintViolations();
        violations.stream().forEach(x ->
errorMsg.append(x.getMessage()).append(";"));
        log.error("[ConstraintViolationException]" +
errorMsg.toString());
        return Result.error(ResultCode.BAD_REQUEST,
errorMsg.toString());
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    @ResponseBody
    public Result handler(MethodArgumentNotValidException e) {
        StringBuffer errorMsg = new StringBuffer();
        List<ObjectError> errors =
e.getBindingResult().getAllErrors();
        errors.stream().forEach(x ->
errorMsg.append(x.getDefaultMessage()).append(";"));
        log.error("[MethodArgumentNotValidException]" +
errorMsg.toString());
        return Result.error(ResultCode.BAD_REQUEST,
errorMsg.toString());
    }

    @ExceptionHandler(BindException.class)
    @ResponseBody
    public Result handler(BindException e) {
        StringBuffer errorMsg = new StringBuffer();
        List<ObjectError> errors = e.getAllErrors();
        errors.stream().forEach(x ->
errorMsg.append(x.getDefaultMessage()).append(";"));
        log.error("[BindException]" + errorMsg.toString());
        return Result.error(ResultCode.BAD_REQUEST,
errorMsg.toString());
    }

}
```

在com.itheima.msg.web.controller中编写：

```
@Slf4j
@RestController
@RequestMapping("mail")
```

```
@Api(tags = "发送邮箱信息")
@Validated
public class MailMsgController {

    @Autowired
    private MailMsgService mailMsgService;

    @ApiImplicitParams({
            @ApiImplicitParam(name = "appName", value = "用户主键
ID", required = true),
            @ApiImplicitParam(name = "sync", value = "sync同步,
async异步", required = true),
            @ApiImplicitParam(name = "mail", value = "目标邮箱地
址", required = true),
            @ApiImplicitParam(name = "code", value = "验证码",
required = true)
    })
    @ApiOperation(value = "发送邮箱验证码")
    @PostMapping(value = "code/{sync}/{appName}")
    public Result sendCode(@PathVariable String appName,
@PathVariable String sync,
                           @Email @NotBlank(message = "邮
箱不能为空") String mail,
                           @Length(min = 6, max = 6,
message = "验证码必须为6位")
                           @NotBlank(message = "验证码不能
为空") String code) {
        log.debug("发送邮箱验证码:appName={},sync={},mail={},code=
{}", appName, sync, mail, code);
        return mailMsgService.sendCode(appName, sync, mail,
code);
    }


    @ApiImplicitParams({
            @ApiImplicitParam(name = "appName", value = "用户主键
ID", required = true),
            @ApiImplicitParam(name = "sync", value = "sync同步,
async异步", required = true),
            @ApiImplicitParam(name = "mail", value = "目标邮箱地
址", required = true),
            @ApiImplicitParam(name = "subject", value = "通知主
题", required = true),
            @ApiImplicitParam(name = "content", value = "通知内
容", required = true)
```

```
    })
    @ApiOperation(value = "发送邮箱通知")
    @PostMapping(value = "notify/{sync}/{appName}")
    public Result sendNotify(@PathVariable String appName,
@PathVariable String sync,
                                        @Email @NotBlank(message =
"邮箱不能为空") String mail,
                                        @NotBlank(message = "通知主题
不能为空") String subject,
                                        @NotBlank(message = "通知内容
不能为空") String content) {
        log.debug("发送邮箱通知:appName={},sync={},mail={},subject=
{},content={}"
                , appName, sync, mail, subject, content);
        return mailMsgService.sendNotify(appName, sync, mail,
subject, content);
    }

    @ApiImplicitParams({
            @ApiImplicitParam(name = "appName", value = "用户主键
ID", required = true),
            @ApiImplicitParam(name = "sync", value = "sync同步,
async异步", required = true),
    })
    @ApiOperation(value = "发送节日礼包提醒")
    @PostMapping(value = "holiday/{sync}/{appName}")
    public Result sendHoliday(@PathVariable String appName,
@PathVariable String sync) {
        log.debug("发送节日礼包提醒: appName={},sync={}", appName,
sync);
        return mailMsgService.sendHoliday(appName, sync);
    }
}
```

6.3 配置文件

在nacos配置文件中修改配置，内容给如下：

```
aliyun:
  accessKey: LTAI4FpSmghKjkTkPeS55cwc
  accessKeySecret: Y6OWohY13Bdh8uBoRukecQus1BhNOf
  maxRequestsPerHost: 20 #每个Host最大连接数
  mail:
```

```
    #应用名称
    tensquare:
      #触发邮件的发件地址
      triggerName: admin@mail.xuziyi.top
      #触发邮件的发件昵称
      triggerAlias: 管理员
      #批量邮件的发件地址
      batchName: msg@mail.xuziyi.top
      #批量邮件节日礼包收件人列表名称
      receiversHoliday: 节日礼包Mail列表
      #批量邮件节日礼包模板
      templateHoliday: 节日礼包
```

在com.itheima.msg.web.util中编写MsgProp，实现加载配置：

```
@Getter
@Setter
@Component
@ConfigurationProperties(prefix = "aliyun")
public class MsgProp {
    private Map<String, Map<String, String>> mail;
}
```

6.4 实现Service

在MailMsgService中编写接口方法：

```
Result sendCode(String appName, String sync, String mail, String
code);

Result<Result> sendNotify(String appName, String sync,
                          String mail, String subject, String
content);

Result<Result> sendHoliday(String appName, String sync);
```

在MailMsgServiceImpl中实现接口方法：

```
@Autowired
private MsgProp msgProp;

@Qualifier("aliMailMsgThread")
@Autowired
private MsgThread<AliMailMsgDto> msgThread;

@Override
public Result sendCode(String appName, String sync, String mail,
String code) {
    //判断该应用是否存在
    if (msgProp.getMail().get(appName) == null) {
        return Result.error(appName + "应用的邮件信息不存在");
    }
    Map<String, String> map = msgProp.getMail().get(appName);

    //封装数据
    AliMailMsgDto dto = AliMailMsgDto.builder()
            .appName(appName)
            .triggerName(map.get("triggerName"))
            .triggerAlias(map.get("triggerAlias"))
            .address(mail)
            .subject(appName + "邮件验证码")
            .content("您的邮件验证码为: " + code)
            .build();

    //发送消息
    if ("sync".equals(sync)) {
        //同步
        return msgThread.send(dto);
    } else if ("async".equals(sync)) {
        //异步
        return msgThread.asyncSend(dto);
    } else {
        return Result.error("同步/异步选择错误");
    }
}

@Override
public Result sendNotify(String appName, String sync,
                         String mail, String subject, String
content) {
    //判断该应用是否存在
    if (msgProp.getMail().get(appName) == null) {
        return Result.error(appName + "应用的邮件信息不存在");
```

```
    }
    Map<String, String> map = msgProp.getMail().get(appName);

    //封装数据
    AliMailMsgDto dto = AliMailMsgDto.builder()
            .appName(appName)
            .triggerName(map.get("triggerName"))
            .triggerAlias(map.get("triggerAlias"))
            .address(mail)
            .subject(subject)
            .content(content)
            .build();

    //发送消息
    if ("sync".equals(sync)) {
        //同步
        return msgThread.send(dto);
    } else if ("async".equals(sync)) {
        //异步
        return msgThread.asyncSend(dto);
    } else {
        return Result.error("同步/异步选择错误");
    }
}

@Override
public Result sendHoliday(String appName, String sync) {
    //判断该应用是否存在
    if (msgProp.getMail().get(appName) == null) {
        return Result.error(appName + "应用的邮件信息不存在");
    }
    Map<String, String> map = msgProp.getMail().get(appName);

    //封装数据
    AliMailMsgDto dto = AliMailMsgDto.builder()
            .appName(appName)
            .batchName(map.get("batchName"))
            .receivers(map.get("receiversHoliday"))
            .template(map.get("templateHoliday"))
            .build();
    ;

    //发送消息
    if ("sync".equals(sync)) {
        //同步
```

```
            return msgThread.sendBath(dto);
        } else if ("async".equals(sync)) {
            //异步
            return msgThread.asyncSendBath(dto);
        } else {
            return Result.error("同步/异步选择错误");
        }
    }
```

7. 实现短信发送

7.1 阿里短信服务

7.1.1 介绍

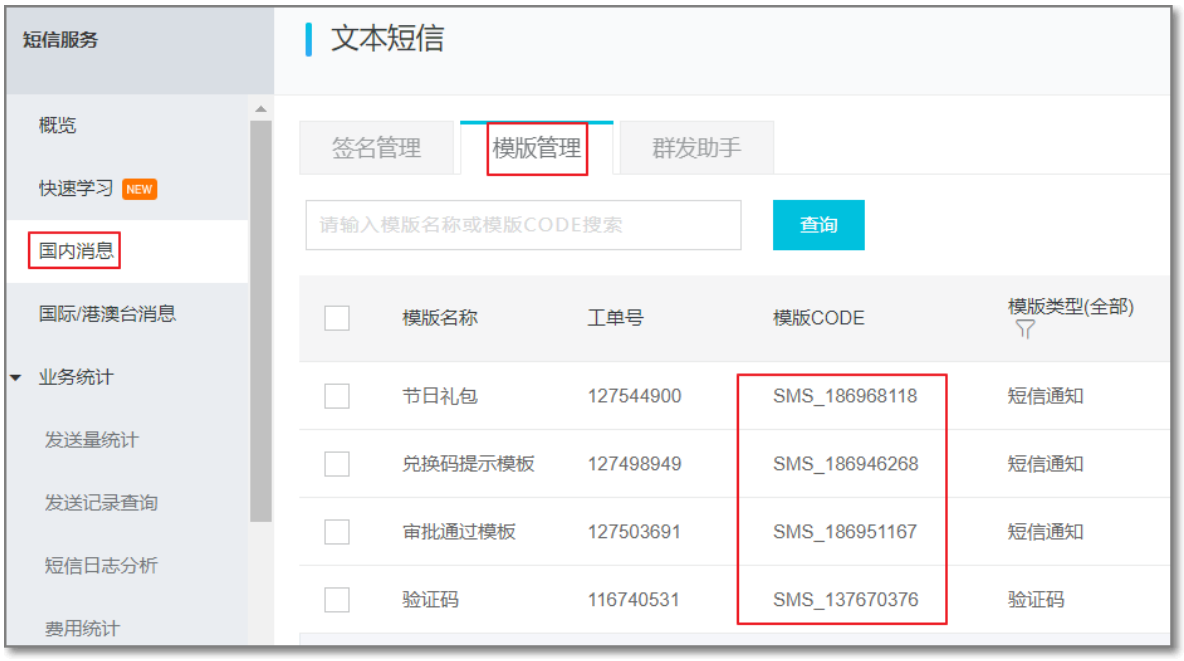
短信服务（Short Message Service）是阿里云为用户提供的一种通信服务的能力。支持国内和国际快速发送验证码、短信通知和推广短信，服务范围覆盖全球200多个国家和地区。
国内短信支持三网合一专属通道，与工信部携号转网平台实时互联。电信级运维保障，实时监控自动切换，
到达率高达99%。完美支撑双11期间20亿短信发送，6亿用户触达。

使用场景如下图：

7.1.2 准备短信资源

申请签名：

申请模板：



需要记录签名和模板CODE

7.2 入门案例

我们使用SDK调用短息服务发送短信

参考网址：https://help.aliyun.com/document_detail/101874.html

SDK	安装说明	DEMO
Java SDK	安装Java SDK	Java SDK DEMO
.NET SDK	安装.NET SDK	.NET SDK DEMO
PHP SDK	安装PHP SDK	PHP SDK DEMO
Python SDK	安装Python SDK	Python SDK DEMO
Node.js SDK	安装Node.js SDK	Node.js SDK DEMO
Go SDK	安装Go SDK	Go SDK DEMO

短信服务支持通过OpenAPI Explorer快速调用短信服务API。同时在OpenAPI Explorer中提供了多种语言的Demo代码。在左侧填写API 参数后，会在**示例代码**页签中自动同步生成对应SDK的Demo代码。

查看Demo

1. 使用阿里云账号登录OpenAPI Explorer。
2. 在**全部产品**中找到**短信服务**。
3. 单击选择需要查看Demo的API接口。
4. 在左侧参数列中指定参数值，右侧**示例代码**页签中会自动同步生成对应SDK的Demo代码。

发送单条短信：

```
public static void main(String[] args) {
    DefaultProfile profile = DefaultProfile.getProfile(
        "cn-hangzhou", "<accessKeyId>", "<accessSecret>");
    IAcsClient client = new DefaultAcsClient(profile);

    CommonRequest request = new CommonRequest();
    request.setSysMethod(MethodType.POST);
    request.setSysDomain("dysmsapi.aliyuncs.com");
    request.setSysVersion("2017-05-25");
    request.setSysAction("SendSms");
    request.putQueryParameter("RegionId", "cn-hangzhou");
    request.putQueryParameter("PhoneNumbers", "18866668888");
    request.putQueryParameter("SignName", "Tree");
    request.putQueryParameter("TemplateCode", "SMS_137670376");
    request.putQueryParameter("TemplateParam", "
{\"code\":\"112233\"}");
    try {
        CommonResponse response =
client.getCommonResponse(request);
        System.out.println(response.getData());
    } catch (ServerException e) {
        e.printStackTrace();
    } catch (ClientException e) {
        e.printStackTrace();
    }
}
```

返回结果：

```
{
    "Message": "OK",
    "RequestId": "7ED9A8EC-CC11-4E01-A293-CF3D75627F6B",
    "BizId": "447409785848695944^0",
    "Code": "OK"
}
```

批量发送短信：

```
 public static void main(String[] args) {
     DefaultProfile profile = DefaultProfile.getProfile(
         "cn-hangzhou", "<accessKeyId>", "<accessSecret>");
     IAcsClient client = new DefaultAcsClient(profile);

     CommonRequest request = new CommonRequest();
     request.setSysMethod(MethodType.POST);
     request.setSysDomain("dysmsapi.aliyuncs.com");
     request.setSysVersion("2017-05-25");
     request.setSysAction("SendBatchSms");
     request.putQueryParameter("RegionId", "cn-hangzhou");
     request.putQueryParameter("PhoneNumberJson", "
[\"18866668888\",\"18833336666\"]");
     request.putQueryParameter("SignNameJson", "
[\"Tree\",\"Tree\"]");
     request.putQueryParameter("TemplateCode", "SMS_137670376");
     request.putQueryParameter("TemplateParamJson",
                               "[{\"code\":\"112233\"},
{\"code\":\"445566\"}]");
     try {
         CommonResponse response =
client.getCommonResponse(request);
         System.out.println(response.getData());
     } catch (ServerException e) {
         e.printStackTrace();
     } catch (ClientException e) {
         e.printStackTrace();
     }
 }
```

返回结果：

```
{
    "Message": "OK",
    "RequestId": "A42EB1C2-A716-41A9-85FE-3E0CFE129F5B",
    "BizId": "217312685848872194^0",
    "Code": "OK"
}
```

7.3 保存发送结果

7.3.1 编写DTO

在com.itheima.msg.web.dto中编写AliSmsMsgDto：

```
@Data
@Builder
@AllArgsConstructor(access = AccessLevel.PRIVATE)
@NoArgsConstructor
public class AliSmsMsgDto {

    //应用名称
    private String appName;
    //模板编号
    private String templateCode;


    //--------单条信息------------
    //发送人电话
    private String phone;
    //签名
    private String sign;
    //模板参数
    private Map<String, String> params;


    //--------批量信息------------
    //发送人电话
    private String[] batchPhone;
    //签名
    private String[] batchSign;
    //模板参数
    private List<Map<String, String>> batchParams;
}
```

7.3.2 编写持久层

短信消息表：

Code	Comment	Data Type
id	主键ID	varchar(20)
app_name	应用名称	varchar(20)
msg_type	消息类型	tinyint
msg_phone	消息接收电话	text
template_id	消息模板ID	varchar(20)
content	消息内容	varchar(200)
success	发送状态	tinyint
result_msg	发送结果	varchar(200)
create_time	创建时间	datetime
modified_time	修改时间	datetime

在com.itheima.msg.web.entity中编写SmsMsg：

```
@Getter
@Setter
@TableName("t_sms_msg")
public class SmsMsg extends BaseEntity {
    private static final long serialVersionUID = 1L;

    @ApiModelProperty(value = "主键ID")
    @TableId(value = "id", type = IdType.ASSIGN_ID)
    private String id;

    private String msgPhone;

    private String templateId;

    private String content;
```

```
    private boolean success;

    private String resultMsg;

}
```

在com.itheima.msg.web.mapper中编写：

```
@Mapper
public interface SmsMsgMapper extends BaseMapper<SmsMsg> {
}
```

7.3.3 编写业务层

在com.itheima.msg.web.service中编写接口，如下：

```
/**
 * <pre>
 * 短信服务 接口
 * </pre>
 */
public interface SmsMsgService extends IService<SmsMsg> {
    void saveResult(AliSmsMsgDto dto, Result result);
}
```

在com.itheima.msg.web.service.impl中实现接口，如下：

```
@Slf4j
@Service
public class SmsMsgServiceImpl extends ServiceImpl<SmsMsgMapper,
SmsMsg>
        implements SmsMsgService {

    @Override
    public void saveResult(AliSmsMsgDto dto, Result result) {
        SmsMsg smsMsg = new SmsMsg();

        smsMsg.setAppName(dto.getAppName());
        smsMsg.setMsgType(MsgType.ALI_YUN.getCode());
```

```
        smsMsg.setTemplateId(dto.getTemplateCode());

        if (StringUtils.isNotBlank(dto.getPhone())) {
            //单条消息
            smsMsg.setMsgPhone(dto.getPhone());

 smsMsg.setContent(JSONUtil.parseFromMap(dto.getParams()).toStrin
g());
        } else {
            //批量消息

 smsMsg.setMsgPhone(JSONUtil.parseArray(dto.getBatchPhone()).toSt
ring());

 smsMsg.setContent(JSONUtil.parseArray(dto.getBatchParams()).toSt
ring());
        }

        //结果信息
        smsMsg.setSuccess(result.getStatus() == ResultCode.OK);
        smsMsg.setResultMsg(result.getData() != null ?
result.getData().toString() : "");

        save(smsMsg);
    }
}
```

7.4 实现发送短信

在中com.itheima.msg.web.sender，编写AliSmsMsgSender，内容如下：

```
@Slf4j
@Component
public class AliSmsMsgSender implements MsgSender<AliSmsMsgDto> {

    @Value("${aliyun.accessKey}")
    private String accessKey;
    @Value("${aliyun.accessKeySecret}")
    private String accessKeySecret;
    @Value("${aliyun.maxRequestsPerHost}")
    private int maxRequestsPerHost;

    @Autowired
```

```
    private SmsMsgService smsMsgService;

    /**
     * 阿里云SDK调用客户端
     */
    private volatile IAcsClient client;

    @Override
    public Result send(AliSmsMsgDto msgData) {
        //声明返回结果对象
        Result result = Result.builder().build();

        try {
            //获取发送请求对象
            CommonRequest request = makeRequest();
            request.setSysAction("SendSms");
            request.putQueryParameter("PhoneNumbers",
msgData.getPhone());
            request.putQueryParameter("TemplateCode",
msgData.getTemplateCode());
            request.putQueryParameter("SignName",
msgData.getSign());

            //设置模板数据
            if (msgData.getParams() != null &&
msgData.getParams().size() > 0) {
                request.putQueryParameter("TemplateParam",

 JSONUtil.parseFromMap(msgData.getParams()).toString());
            }

            //发起请求获取响应
            CommonResponse response =
getClient().getCommonResponse(request);

            //解析响应，判断发送结果
            Map map = JSONUtil.toBean(response.getData(),
Map.class);
            if ("OK".equals(map.get("Code"))) {
                result.setStatus(ResultCode.OK);
                result.setMsg("手机信息发送成功");
            } else {
                result.setStatus(ResultCode.ERROR);
                result.setMsg("手机信息发送失败");
            }
```

```
            result.setData(response.getData());

        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
            result.setStatus(ResultCode.ERROR);
            result.setMsg("手机发送失败，程序异常");
            result.setData(e.getMessage());
        }


        try {
            //保存信息数据
            smsMsgService.saveResult(msgData, result);
        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
        }

        //返回结果对象
        return result;
    }

    @Override
    public Result batchSend(AliSmsMsgDto msgData) {
        //声明返回结果对象
        Result result = Result.builder().build();

        try {
            //获取发送请求对象
            CommonRequest request = makeRequest();
            request.setSysAction("SendBatchSms");
            request.putQueryParameter("TemplateCode",
msgData.getTemplateCode());
            request.putQueryParameter("PhoneNumberJson",

 JSONUtil.parseArray(msgData.getBatchPhone()).toString());
            request.putQueryParameter("SignNameJson",

 JSONUtil.parseArray(msgData.getBatchSign()).toString());

            //设置模板数据
            if (msgData.getBatchParams() != null &&
msgData.getBatchParams().size() > 0) {
                request.putQueryParameter("TemplateParamJson",

 JSONUtil.parseArray(msgData.getBatchParams()).toString());
```

```
            }

            //发起请求获取响应
            CommonResponse response =
getClient().getCommonResponse(request);

            //解析响应，判断发送结果
            Map map = JSONUtil.toBean(response.getData(),
Map.class);
            if ("OK".equals(map.get("Code"))) {
                result.setStatus(ResultCode.OK);
                result.setMsg("手机信息批量发送成功");
            } else {
                result.setStatus(ResultCode.ERROR);
                result.setMsg("手机信息批量发送失败");
            }
            result.setData(response.getData());

        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
            result.setStatus(ResultCode.ERROR);
            result.setMsg("手机信息批量发送失败，程序异常");
            result.setData(e.getMessage());
        }


        try {
            //保存信息数据
            smsMsgService.saveResult(msgData, result);
        } catch (Exception e) {
            log.error(ExceptionUtils.getStackTrace(e));
        }

        //返回结果对象
        return result;
    }

    private CommonRequest makeRequest() {
        CommonRequest request = new CommonRequest();
        request.setSysMethod(MethodType.POST);
        request.setSysDomain("dysmsapi.aliyuncs.com");
        request.setSysVersion("2017-05-25");
        request.putQueryParameter("RegionId", "cn-hangzhou");

        return request;
```

```
    }

    /**
     * 获取阿里云短信平台客户端
     */
    private IAcsClient getClient() {
        if (client == null) {
            synchronized (AliSmsMsgSender.class) {
                if (client == null) {
                    DefaultProfile profile =
DefaultProfile.getProfile("cn-hangzhou"
                            , accessKey, accessKeySecret);

                    HttpClientConfig config =
HttpClientConfig.getDefault();

 config.setMaxRequestsPerHost(maxRequestsPerHost);
                    config.setConnectionTimeoutMillis(10000L);

                    profile.setHttpClientConfig(config);
                    client = new DefaultAcsClient(profile);
                }
            }
        }

        return client;
    }
}
```

7.5 实现多线程调用

在com.itheima.msg.web.thread中编写AliSmsMsgThread，内容如下：

```
/**
 * <pre>
 * 消息发送服务线程
 * </pre>
 */
@Slf4j
@Component
public class AliSmsMsgThread extends BaseMsgThread implements
MsgThread<AliSmsMsgDto> {
```

```
    @Qualifier("aliSmsMsgSender")
    @Autowired
    private MsgSender msgSender;

    @Override
    public Result send(AliSmsMsgDto msgData) {
        AtomicReference<Result> result = new AtomicReference<>();
        //计数闭锁
        CountDownLatch countDownLatch = new CountDownLatch(1);
        getPool().execute(() -> {
            result.set(msgSender.send(msgData));
            //计数闭锁计数
            countDownLatch.countDown();
        });

        //计数闭锁等待
        try {
            countDownLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        return result.get();
    }

    @Override
    public Result sendBath(AliSmsMsgDto msgData) {
        AtomicReference<Result> result = new AtomicReference<>();
        //计数闭锁
        CountDownLatch countDownLatch = new CountDownLatch(1);
        getPool().execute(() -> {
            result.set(msgSender.bathSend(msgData));
            //计数闭锁计数
            countDownLatch.countDown();
        });


        //计数闭锁等待
        try {
            countDownLatch.await();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        return result.get();
```

```
    }

    @Override
    public Result asyncSend(AliSmsMsgDto msgData) {
        getPool().execute(() -> msgSender.send(msgData));
        return Result.ok("已经执行异步消息发送", null);
    }

    @Override
    public Result asyncSendBath(AliSmsMsgDto msgData) {
        getPool().execute(() -> msgSender.bathSend(msgData));
        return Result.ok("已经执行异步消息发送", null);
    }
}
```

8. 实现短信API接口

8.1 编写手机校验注解

在com.itheima.msg.web.validate中编写ValidUtil，如下：

```
public class ValidUtil {

    //手机号校验正则
    public static final String MOBILE_REGX = "^[1][3-9][0-9]
{9}$";
    public static final String MOBILE_MSG = "手机号格式错误";

}
```

在com.itheima.msg.web.validate中编写Mobile注解，如下：

```
@Target({METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR, PARAMETER,
TYPE_USE})
@Retention(RUNTIME)
//用于校验手机号的逻辑类
@Constraint(validatedBy = MobileValidator.class)
public @interface Mobile {
    //手机号的校验格式
    String regexp() default ValidUtil.MOBILE_REGX;
```

```
    //出现错误返回的信息
    String message() default ValidUtil.MOBILE_MSG;

    Class<?>[] groups() default {};

    Class<? extends Payload>[] payload() default {};

    @Target({METHOD, FIELD, ANNOTATION_TYPE, CONSTRUCTOR,
PARAMETER, TYPE_USE})
    @Retention(RUNTIME)
    @Documented
    public @interface List {
        Mobile[] value();
    }
}
```

在com.itheima.msg.web.validate中编写，如下：

```
public class MobileValidator implements
ConstraintValidator<Mobile, String> {

    private String regexp;

    //初始化方法
    @Override
    public void initialize(Mobile constraintAnnotation) {
        //获取校验的手机号的格式
        this.regexp = constraintAnnotation.regexp();
    }

    @Override
    public boolean isValid(String value,
ConstraintValidatorContext context) {
        if(value == null){
            return true;
        }

        return value.matches(regexp);
    }
}
```

8.2 编写Controller

我们要完成2个接口

POST /sms/code/{sync}/{appName} 发送短信验证码

POST /sms/holiday/{sync}/{appName} 发送节日礼包提醒

请求路径第一个参数使用 sync--同步处理 async--异步处理

请求路径第二个参数appName，传递不同应用的应用名，以方便管理

在com.itheima.msg.web.controller中编写SmsMsgController，如下：

```
@Slf4j
@RestController
@RequestMapping("sms")
@Api(tags = "发送手机短信")
@Validated
public class SmsMsgController {

    @Autowired
    private SmsMsgService smsMsgService;

    //POST   /sms/code/{sync}/{appName}      发送短信验证码
    @ApiImplicitParams({
            @ApiImplicitParam(name = "sync", value = "sync同步,
async异步模式", required = true),
            @ApiImplicitParam(name = "appName", value = "用户应用名
称", required = true),
            @ApiImplicitParam(name = "phone", value = "发送目标的手
机号", required = true),
            @ApiImplicitParam(name = "code", value = "验证码",
required = true),
    })
    @ApiOperation(value = "发送手机验证码")
    @PostMapping(value = "code/{sync}/{appName}")
    public Result sendCode(@PathVariable String sync,
@PathVariable String appName,
                           @Mobile @NotBlank(message = "手机号不能
为空") String phone,
                           @NotBlank(message = "验证码不能为空")
String code) {
```

```
        log.debug("发送手机验证码:sync={},appName={},phone={},code={}", sync, appName, phone, code);
        return smsMsgService.sendCode(appName, sync, phone, code);
    }

    //POST   /sms/holiday/{sync}/{appName}  发送节日礼包提醒
    @ApiImplicitParams({
            @ApiImplicitParam(name = "sync", value = "sync同步,async异步模式", required = true),
            @ApiImplicitParam(name = "appName", value = "用户应用名称", required = true),
            @ApiImplicitParam(name = "phone", value = "发送目标的手机号,多个手机号用,分隔", required = true),
            @ApiImplicitParam(name = "name", value = "用户名称，多个用户名称用,分隔", required = true),
    })
    @ApiOperation(value = "批量发送节日礼包提醒")
    @PostMapping(value = "holiday/{sync}/{appName}")
    public Result sendHoliday(@PathVariable String sync, @PathVariable String appName,
                              @NotBlank(message = "手机号不能为空") String phone,
                              @NotBlank(message = "用户名称不能为空") String name) {
        log.debug("批量发送节日礼包提醒：sync={},appName={},phone={},name={}", sync, appName, phone, name);

        return smsMsgService.sendHoliday(sync, appName, phone, name);
    }
}
```

8.3 添加配置

在nacos中修改配置，内容如下：

```yaml
aliyun:
  。。。。。。
  sms:
    #应用名称
    tensquare:
      #签名名称
      signName: Tree
      #发送验证码
      templateCode: SMS_137670376
      #兑换码通知
      templateHoliday: SMS_186968118
```

修改MsgProp，如下：

```
@Getter
@Setter
@Component
@ConfigurationProperties(prefix = "aliyun")
public class MsgProp {
    private Map<String, Map<String, String>> mail;

    private Map<String, Map<String, String>> sms;
}
```

8.4 编写Service

在中添加接口方法，如下：

```
Result sendCode(String appName, String sync, String phone, String
code);

Result sendHoliday(String appName, String sync, String[]
phoneList, String[] nameList);
```

在中实现接口方法，如下：

```
@Autowired
private MsgProp msgProp;
```

```
@Qualifier("aliSmsMsgThread")
@Autowired
private MsgThread<AliSmsMsgDto> msgThread;

@Override
public Result sendCode(String appName, String sync, String phone,
String code) {
    //判断该应用是否存在
    if (msgProp.getSms().get(appName) == null) {
        return Result.error(appName + "应用的邮件信息不存在");
    }
    Map<String, String> map = msgProp.getSms().get(appName);

    //封装数据
    AliSmsMsgDto dto = AliSmsMsgDto.builder()
            .appName(appName)
            .templateId(map.get("templateCode"))
            .phone(phone)
            .sign(map.get("signName"))
            .params(MapUtil.builder("code", code).build())
            .build();

    //发送消息
    if ("sync".equals(sync)) {
        //同步
        return msgThread.send(dto);
    } else if ("async".equals(sync)) {
        //异步
        return msgThread.asyncSend(dto);
    } else {
        return Result.error("同步/异步选择错误");
    }
}

@Override
public Result sendHoliday(String appName, String sync, String[]
phones, String[] names) {
    //判断该应用是否存在
    if (msgProp.getSms().get(appName) == null) {
        return Result.error(appName + "应用的邮件信息不存在");
    }
    Map<String, String> map = msgProp.getSms().get(appName);

    //准备数据
    String[] signs = new String[names.length];
```

```
    List<Map<String, String>> paramsList = new ArrayList<>();
    for (int i = 0; i < names.length; i++) {
        paramsList.add(MapUtil.builder("name",
names[i]).build());
        signs[i] = map.get("signName");
    }

    //封装数据
    AliSmsMsgDto dto = AliSmsMsgDto.builder()
            .appName(appName)
            .templateId(map.get("templateHoliday"))
            .batchPhone(phones)
            .batchSign(signs)
            .batchParams(paramsList)
            .build();

    //发送消息
    if ("sync".equals(sync)) {
        //同步
        return msgThread.sendBath(dto);
    } else if ("async".equals(sync)) {
        //异步
        return msgThread.asyncSendBath(dto);
    } else {
        return Result.error("同步/异步选择错误");
    }
}
```