邮件推送

API 参考

API 参考

简介

欢迎使用阿里云邮件推送服务 DirectMail。您可以根据本文档介绍的 API 对 DirectMail 服务进行相关操作。

请确保在使用这些接口前，已充分了解 DirectMail 产品说明、使用协议和收费方式。

术语表

术语	中文	说明
AccountName	发信地址	必须先通过管理控制台预先创建一个发信地址。 DirectMail 服务需要对用户的发信地址进行验证。
Template	邮件模板	批量发信需预先创建一个邮件模板。
Receiver	收件人列表	批量发信需预先创建一个收信人列表。

业务资源规格限制说明

DirectMail 对每个用户可拥有的发信地址、邮件模板、收件人列表、每个账号的发信数量等规格方面均有限制。在使用 DirectMail Open API 时，请参考官网上最新的业务限制规则和资源规格限制：DirectMail 产品和业务限制。

在接口说明部分，凡出现对参数可选值、可用规格方面与官网上给出的资源规格限制发生矛盾时，均以官网上给出的值为准。

API概览

发信邮件相关接口

API	描述
SingleSendMail	单一发信接口，支持发送触发和其他单个邮件
BatchSendMail	批量发信接口，支持通过调用模板的方式发送批量邮件

相关文档

- API 资源导航
- API Explorer
- API 错误中心

API服务地址

杭州（华东 1）：
RegionId:cn-hangzhou
Host:dm.aliyuncs.com
Version:2015-11-23

新加坡（新加坡）：
RegionId:ap-southeast-1
Host:dm.ap-southeast-1.aliyuncs.com
Version:2017-06-22

澳洲（悉尼）：
RegionId:ap-southeast-2
Host:dm.ap-southeast-2.aliyuncs.com
Version:2017-06-22

调用方式

概述

通过向 DirectMail API 的服务端地址发送 HTTP 的 GET 或 POST 请求，并按照接口说明在请求中加入相应请求参数来完成对 DirectMail API 接口调用。根据请求的处理情况，系统会返回处理结果。

详细信息，请参见：

- 请求结构
- 公共参数
- 返回结果
- 签名机制

请求结构

服务地址

DirectMail API 的服务接入地址:

- 华东1：dm.aliyuncs.com
- 亚太东南1（新加坡）：dm.ap-southeast-1.aliyuncs.com
- 亚太东南2（悉尼）：dm.ap-southeast-2.aliyuncs.com

通信协议

支持通过 HTTP 或 HTTPS 通道进行请求通信。为了获得更高的安全性，推荐您使用HTTPS 通道发送请求。

请求方法

支持 HTTP GET 方法发送请求，这种方式下请求参数需要包含在请求的 URL 中。支持 HTTP POST 方法发送请求，这种方式下请求参数需要包含在请求的 BODY 中。

请求参数

每个请求都需要指定要执行的操作，即Action参数（例如 SingleSendMail），以及每个操作都需要包含的公

共请求参数和指定操作所特有的请求参数。

字符编码

请求及返回结果都使用UTF-8字符集进行编码。

公共参数

公共请求参数

公共请求参数是指调用每个接口都需要使用到的请求参数。

名称	类型	是否必须	描述
Format	String	否	返回值的类型，支持 JSON 与 XML。默认为 XML。
Version	String	是	API 版本号，为日期形式：YYYY-MM-DD。如果参数 RegionID 是 cn-hangzhou，则版本对应为2015-11-23；如参数 RegionID 是cn-hangzhou 以外其他 Region，比如 ap-southeast-1，则版本对应为2017-06-22。
AccessKeyId	String	是	阿里云颁发给用户的访问服务所用的密钥 ID。
Signature	String	是	签名结果串，关于签名的计算方法，请参见签名机制。
SignatureMethod	String	是	签名方式，目前支持 HMAC-SHA1。
Timestamp	String	是	请求的时间戳。日期格式按照 ISO8601 标准表示，并需要使用 UTC 时间。格式为 YYYY-MM-DDThh:mm:ssZ。 例如，2015-11-23T04:00:00Z（为北

			京时间 2015 年 11 月 23 日 12 点 0 分 0 秒）。
SignatureVersion	String	是	签名算法版本，目前版本是1.0。
SignatureNonce	String	是	唯一随机数，用于防止网络重放攻击。不同的请求要使用不同的随机数值。您可以使用 UUID（随机串），也可以自定义。
RegionId	String	否	机房信息 ，目前支持 cn-hangzhou、ap-southeast-1、ap-southeast-2。

示例

```
https://dm.aliyuncs.com/
?Format=xml
&Version=2015-11-23
&Signature=Pc5WB8gokVn0xfeu%2FZV%2BiNM1dgI%3D
&SignatureMethod=HMAC-SHA1
&SignatureNonce=e1b44502-6d13-4433-9493-69eeb068e955
&SignatureVersion=1.0
&AccessKeyId=key-test
&Timestamp=2015-11-23T12:00:00Z
```

公共返回参数

用户发送的每次接口调用请求，无论成功与否，系统都会返回一个唯一识别码 RequestId。

参数	类型	描述
RequestId	String	阿里云为该请求生成的唯一标识符。

示例

XML示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!—结果的根结点-->
<接口名称+Response>
<!—返回请求标签-->
<RequestId>4C467B38-3910-447D-87BC-AC049166F216</RequestId>
```

```
<!—返回结果数据-->
</接口名称+Response>
```

JSON示例

```
{
"RequestId": "4C467B38-3910-447D-87BC-AC049166F216"
/* 返回结果数据 */
}
```

返回结果

调用API服务后返回数据采用统一格式：

- 返回的 HTTP 状态码为 2xx，代表调用成功；
- 返回的 HTTP 状态码为 4xx 或 5xx 代表调用失败。

调用成功返回的数据格式主要有 XML 和 JSON 两种，外部系统可以在请求时传入参数来制定返回的数据格式，默认为 XML 格式。

> 本文档中的返回示例为了便于用户查看，做了格式化处理，实际返回结果是没有进行换行、缩进等处理的。

成功结果

XML示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!—结果的根结点-->
<接口名称+Response>
<!—返回请求标签-->
<RequestId>4C467B38-3910-447D-87BC-AC049166F216</RequestId>
<!—返回结果数据-->
</接口名称+Response>
```

JSON示例

```
{
"RequestId": "4C467B38-3910-447D-87BC-AC049166F216"
```

```
/* 返回结果数据 */
}
```

错误结果

调用接口出错后，将不会返回结果数据。调用方可根据每个接口对应的错误码以及 错误代码表 来定位错误原因。

当调用出错时，HTTP 请求返回一个 4xx 或 5xx 的 HTTP 状态码。返回的消息体中是具体的错误代码及错误信息。另外还包含一个全局唯一的请求 ID：RequestId 和一个您该次请求访问的站点 ID：HostId。在调用方找不到错误原因时，可以联系阿里云客服，并提供该 HostId 和 RequestId，以便我们尽快帮您解决问题。

XML示例

```
<?xml version="1.0" encoding="UTF-8"?>
<Error>
<RequestId>8906582E-6722-409A-A6C4-0E7863B733A5</RequestId>
<HostId>dm.aliyuncs.com</HostId>
<Code>InvalidTemplate.NotFound</Code>
<Message>The specified template does not found.</Message>
</Error>
```

JSON示例

```
{
"RequestId": "8906582E-6722-409A-A6C4-0E7863B733A5",
"HostId": "dm.aliyuncs.com",
"Code": "InvalidTemplate.NotFound",
"Message": "The specified template does not found."
}
```

签名机制

注意：

- 若使用阿里云提供的 SDK，则无需查看签名机制。当前已提供 Java、PHP、C# 的 SDK。
- 接口支持通过 GET 和 POST 提交请求，但是 GET 和 POST 的 StringToSign 不一样。

邮件推送服务会对每个访问请求进行身份验证，所以无论使用 HTTP 还是 HTTPS 协议提交请求，都需要在请求中包含签名（Signature）信息。邮件推送通过使用 AccessKeyID 和 AccessKeySecret 进行对称加密的方法来验证请求的发送者身份。AccessKeyID 和 AccessKeySecret 由阿里云官方颁发给用户的 AccessKey 信息

（可以通过阿里云控制台**用户信息管理**中查看和管理）。其中 AccessKeyID 用于标识访问者的身份；AccessKeySecret 是用于加密签名字符串和服务器端验证签名字符串的密钥。AccessKey 信息必须严格保密。

签名步骤

在访问时，按照下面的方法对请求进行签名处理：

使用请求参数构造规范化的请求字符串（Canonicalized Query String）

a) 排序参数。

按照参数名称的字典顺序，对请求中所有的请求参数（包括文档中描述的公共请求参数和给定的请求接口的自定义参数，但不包括公共请求参数中的 Signature 参数本身）进行排序。

注意：当使用 GET 方法提交请求时，这些参数就是请求 URI 中的参数部分（即 URI 中？之后由 & 连接的部分）。

b) 对每个请求参数的名称和值进行编码。

参数名称和值要使用 UTF-8 字符集进行 URL 编码。URL 编码的编码规则是：

- 对于字符 A-Z、a-z、0-9 以及字符 -、_、.、~ 不编码。
- 对于其他字符编码成 %XY 的格式，其中 XY 是字符对应 ASCII 码的 16 进制表示。比如半角的双引号 " 对应的编码就是 %22。
- 对于扩展的 UTF-8 字符，编码成 %XY%ZA... 的格式。
- 需要说明的是半角的空格要被编码是 %20，而不是加号 +。

注意：一般支持 URL 编码的库（比如 Java 中的 java.net.URLEncoder）都是按照 application/x-www-form-urlencoded 的 MIME 类型的规则进行编码。可以直接使用这类方式进行编码，把编码后的字符串中加号 + 替换成 %20、星号 * 替换成 %2A、%7E 替换回波浪号 ~，即可得到上述规则描述的编码字符串。

c) 使用半角的等号 = 连接编码后的参数名称和参数值。

d) 使用 & 符号连接编码后的请求参数（参数排序与步骤 a 的排序一致），即得到规范化请求字符串。

使用上一步构造的规范化字符串，按照下面的规则构造用于计算签名的字符串:

```
 StringToSign=
HTTPMethod + "&" +
percentEncode("/") + "&" +
percentEncode(CanonicalizedQueryString)
```

其中 HTTPMethod 是提交请求用的 HTTP 方法，比如 GET 和 POST。

percentEncode(“/”)是按照 1.b 中描述的 URL 编码规则对字符 /进行编码得到的值，即 %2F。

percentEncode (CanonicalizedQueryString) 是对第 1 步中构造的规范化请求字符串，再次按 1.b 中描述的 URL 编码规则进行编码后得到的字符串。**注意：这里的 percentEncode (CanonicalizedQueryString) 是两次编码后得到的结果。**

3. 计算 HMAC 值。按照 RFC2104 的定义，使用步骤 2 得到的字符串 StringToSign 计算签名 HMAC 值。**注意：**计算签名时，使用的 Key 就是您的 AccessKeySecret 加上一个 & 字符 (ASCII:38)，使用的哈希算法是 SHA1。
4. 计算签名值。按照 Base64 编码规则，把步骤 3 得到的 HMAC 值编码成字符串，即得到签名值（Signature）。
5. 将得到的签名值作为 Signature 参数添加到请求参数中，即完成请求签名过程。**注意：**得到的签名值作为请求参数值提交给邮件推送服务器的时候，要和其他参数一样，按照 RFC3986 的规则进行 URL 编码）。

签名示例

以 SingleSendMail 接口，通过 HTTPS 发送 POST 请求调用说明为例：

请求URL为：http://dm.aliyuncs.com/ 。

请求参数为：

```
AccessKeyId=testid&AccountName=<a%b'>&Action=SingleSendMail&AddressType=1&Format=XML&HtmlBody=4&RegionId=cn-hangzhou&ReplyToAddress=true&SignatureMethod=HMAC-SHA1&SignatureNonce=c1b2c332-4cfb-4a0f-b8cc-ebe622aa0a5c&SignatureVersion=1.0&Subject=3&TagName=2&Timestamp=2016-10-20T06:27:56Z&ToAddress=1@test.com&Version=2015-11-23
```

那么 StringToSign 就是

```
POST&%2F&AccessKeyId%3Dtestid%26AccountName%3D%253Ca%2525b%2527%253E%26Action%3DSingleSendMail%26AddressType%3D1%26Format%3DXML%26HtmlBody%3D4%26RegionId%3Dcn-hangzhou%26ReplyToAddress%3Dtrue%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3Dc1b2c332-4cfb-4a0f-b8cc-ebe622aa0a5c%26SignatureVersion%3D1.0%26Subject%3D3%26TagName%3D2%26Timestamp%3D2016-10-20T06%253A27%253A56Z%26ToAddress%3D1%2540test.com%26Version%3D2015-11-23
```

假如使用的 AccessKeyId 是 testid，AccessKeySecret 是 testsecret，用于计算 HMAC 的 Key 就是 testsecret&，则计算得到的签名值是：

```
lIJfXJjBW3OacrVgxxsITgYaYm0=
```

签名后，请求 http://dm.aliyuncs.com/ 发起 POST 请求的 BODY内容参数如下（增加了 Signature 参数和修改了请求头 Content-Type: application/x-www-form-urlencoded）：

```
Signature=lIJfXJjBW3OacrVgxxsITgYaYm0=&AccessKeyId=testid&AccountName=<a%b'>&Action=SingleSendMail
&AddressType=1&Format=XML&HtmlBody=4&RegionId=cn-
hangzhou&ReplyToAddress=true&SignatureMethod=HMAC-SHA1&SignatureNonce=c1b2c332-4cfb-4a0f-b8cc-
ebe622aa0a5c&SignatureVersion=1.0&Subject=3&TagName=2&Timestamp=2016-10-
20T06:27:56Z&ToAddress=1@test.com&Version=2015-11-23
```

以上提供的是编码之前的BODY所需参数，具体请求需要对参数进行编码，详情请参考请求示例

相关代码参考：

java版

```
private static final String MAC_NAME ="HmacSHA1";
private static final String ENCODING ="UTF-8";

/**
* 使用 HMAC-SHA1 签名方法对对encryptText进行签名
*
*@param encryptText 被签名的字符串
*@param encryptKey 密钥
*@return
*@throws Exception
*/
public static byte[] HmacSHA1Encrypt(StringencryptText, String encryptKey) throwsException {
byte[] data = encryptKey.getBytes(ENCODING);
//根据给定的字节数组构造一个密钥,第二参数指定一个密钥算法的名称
SecretKey secretKey = new SecretKeySpec(data,MAC_NAME);
//生成一个指定 Mac 算法的 Mac 对象
Mac mac = Mac.getInstance(MAC_NAME);
//用给定密钥初始化 Mac 对象
mac.init(secretKey);

byte[] text = encryptText.getBytes(ENCODING);
//完成 Mac 操作
return mac.doFinal(text);
}

@Test
public void test() throws Exception {
/*StringToSign=
HTTPMethod + “&” +
percentEncode(“/”) + ”&” +
percentEncode(CanonicalizedQueryString*/

String str
="AccessKeyId=testid&AccountName=<a%b'>&Action=SingleSendMail&AddressType=1&Format=XML&HtmlBod
y=4&RegionId=cn-hangzhou&ReplyToAddress=true&SignatureMethod=HMAC"+
"-SHA1&SignatureNonce=c1b2c332-4cfb-4a0f-b8cc-
ebe622aa0a5c&SignatureVersion=1.0&Subject=3&TagName=2&Timestamp=2016-10-
20T06:27:56Z&ToAddress=1@test.com&Version=2015-11-23";
```

```
String percentStr = "";
String[] strs = str.split("&");
for (int i = 0; i < strs.length;i++) {
String[] str1 =strs[i].split("=");

if (str1.length == 1){
percentStr = percentStr + getUtf8Encoder(str1[0])+ "=" + getUtf8Encoder("")+ "&";
}else {
percentStr = percentStr + getUtf8Encoder(str1[0])+ "=" + getUtf8Encoder(str1[1]) + "&";
}
}
percentStr =percentStr.substring(0,percentStr.lastIndexOf("&"));

String percent = URLEncoder.encode("/","UTF-8");
percentStr = getUtf8Encoder(percentStr);

String toSign = HttpMethod.POST + "&"+ percent +"&" + percentStr;
System.out.println("--------------"+ toSign);

/*POST&%2F&AccessKeyId%3Dtestid%26AccountName%3D%253Ca%2525b%2527%253E%26Action%3DSingleSendMail%26AddressType%3D1%26Format%3DXML%26HtmlBody%3D4%26RegionId%3Dcn-hangzhou%26ReplyToAddress%3Dtrue%26SignatureMethod%3DHMAC-SHA1%26SignatureNonce%3Dc1b2c332-4cfb-4a0f-b8cc-ebe622aa0a5c%26SignatureVersion%3D1.0%26Subject%3D3%26TagName%3D2%26Timestamp%3D2016-10-20T06%253A27%253A56Z%26ToAddress%3D1%2540test.com%26Version%3D2015-11-23*/


byte[] bytes = HmacSHA1Encrypt(toSign,"testsecret&");
String base64Str = Base64.encode(bytes);
System.out.println(base64Str);//llJfXJjBW3OacrVgxxsITgYaYm0=
}

private String getUtf8Encoder(String param) throwsUnsupportedEncodingException {
return URLEncoder.encode(param, "UTF-8")
.replaceAll("\\+","%20")
.replaceAll("\\*","%2A")
.replaceAll("%7E","~");
};
```

Python版

```
import hmac
import base64
#其中StringToSign需要替换为对应的内容。
message = b'StringToSign'
key = b'testsecret&'
h = hmac.new(key, message, digestmod='sha1')
# print(h.digest())
print(base64.b64encode(h.digest()))
```

PHP版

```
<?php
function encryptTokey($data){
$apikey = 'testsecret&';
$sign = base64_encode(hash_hmac("sha1", $data, $apikey, true));
//printf($sign);
return $sign;
}
//其中StringToSign需要替换为对应的内容。
echo encryptTokey(StringToSign);
?>
```

请求示例

API请求调用相关示例

```
import com.sun.org.apache.xerces.internal.impl.dv.util.Base64;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpMethod;
import org.testng.annotations.Test;

import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import javax.net.ssl.*;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLConnection;
import java.net.URLEncoder;
import java.security.cert.X509Certificate;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.UUID;

public class InnerHttpRequest {
private static final Logger logger = LoggerFactory.getLogger(InnerHttpRequest.class);
final static HostnameVerifier DO_NOT_VERIFY = new HostnameVerifier() {
public boolean verify(String hostname, SSLSession session) {
return true;
}
};

private static void trustAllHosts() {
```

```
// Create a trust manager that does not validate certificate chains
TrustManager[] trustAllCerts = new TrustManager[]{new X509TrustManager() {
public java.security.cert.X509Certificate[] getAcceptedIssuers() {
return new java.security.cert.X509Certificate[]{};
}

public void checkClientTrusted(X509Certificate[] chain, String authType) {

}

public void checkServerTrusted(X509Certificate[] chain, String authType) {

}
}};

// Install the all-trusting trust manager
try {
SSLContext sc = SSLContext.getInstance("TLS");
sc.init(null, trustAllCerts, new java.security.SecureRandom());
HttpsURLConnection.setDefaultSSLSocketFactory(sc.getSocketFactory());
} catch (Exception e) {
e.printStackTrace();
}
}

public String doHttpRequest(String url, String param) {
trustAllHosts();
String result = "";
BufferedReader in = null;
try {
String urlNameString = url + "?" + param;
URL realUrl = new URL(urlNameString);
URLConnection connection = realUrl.openConnection();

connection.setRequestProperty("accept", "*/*");
connection.setRequestProperty("connection", "Keep-Alive");
connection.setRequestProperty("user-agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");


if (realUrl.getProtocol().toLowerCase().equals("https")) {
HttpsURLConnection https = (HttpsURLConnection) connection;
https.setHostnameVerifier(DO_NOT_VERIFY);
https.setRequestMethod(HttpMethod.GET.toString());
https.connect();
in = new BufferedReader(new InputStreamReader(https.getInputStream()));
} else {
connection.connect();
in = new BufferedReader(new InputStreamReader(connection.getInputStream()));
}

String line;
while ((line = in.readLine()) != null) {
result += line;
}
} catch (Exception e) {
logger.info("InnerHttpRequest doHttpRequest发送GET请求出现异常！" + e);
```

```
e.printStackTrace();
} finally {
try {
if (in != null) {
in.close();
}
} catch (Exception e2) {
e2.printStackTrace();
}
}
return result;
}

public String doPostHttpRequest(String url, String param) {
trustAllHosts();
String result = "";
BufferedReader in = null;
try {
String urlNameString = url + "?" + param;
URL realUrl = new URL(urlNameString);
URLConnection connection = realUrl.openConnection();

connection.setRequestProperty("accept", "*/*");
connection.setRequestProperty("connection", "Keep-Alive");
connection.setRequestProperty("user-agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;SV1)");
connection.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
connection.setRequestProperty("Content-Length", String.valueOf(param.length()));

if (realUrl.getProtocol().toLowerCase().equals("https")) {
HttpsURLConnection https = (HttpsURLConnection) connection;
https.setHostnameVerifier(DO_NOT_VERIFY);
https.setRequestMethod(HttpMethod.POST.toString());
https.connect();
in = new BufferedReader(new InputStreamReader(https.getInputStream()));

} else {
HttpURLConnection http = (HttpURLConnection) connection;
http.setRequestMethod(HttpMethod.POST.toString());
http.connect();
in = new BufferedReader(new InputStreamReader(http.getInputStream()));
}

String line;
while ((line = in.readLine()) != null) {
result += line;
}
} catch (Exception e) {
logger.info("InnerHttpRequest doHttpRequest发送GET请求出现异常！" + e);
e.printStackTrace();
} finally {
try {
if (in != null) {
in.close();
}
} catch (Exception e2) {
e2.printStackTrace();
```

```
}
}
return result;
}

public String prepareParamStrURLEncoder(String[] keys, Object[] values) {
StringBuffer params = new StringBuffer();
String param = "";
try {
for (int i = 0; i < keys.length; i++) {
logger.info(keys[i] + ":" + values[i]);
if (values[i].equals("") || values[i] == null) {
continue;
}
params.append(getUtf8Encoder(keys[i]) + "=" + getUtf8Encoder(values[i].toString()) + "&");

}
param = params.substring(0, params.lastIndexOf("&"));

} catch (Exception e) {
logger.info("InnerHttpRequest prepareParamStr(String[] keys, String[] values)转换url编码异常！" + e);
e.printStackTrace();
}
System.out.println("-----prepareParamStrURLEncoder-----" + param);
return param;
}

private final static String protocol = "https";
private final static String host = "dm.aliyuncs.com";
private final static String AccessKeyId = "";
private final static String AccessKeySecret = "";
private final static String AccountName = " noreply@***.club";
private final static String Format = "JSON";
private final static String ToAddress = " noreply_test@***.club";
private final static String SignatureMethod = "HMAC-SHA1";
private final static String SignatureVersion = "1.0";
private final static String Version = "2015-11-23";
private final static String AddressType = "1";
private final static String RegionId = "cn-hangzhou";
private final static Boolean ReplyToAddress = Boolean.TRUE;
private final static String HtmlBody = "<html><body><h3>Test send to email</h3></body></html> <a%b' + *
%7E> （ ）！测试邮件正文。你此次申请注册的验证码为：123456";
private final static String Subject = "测试主题";
private final static String TagName = "测试Tag";
private final static HttpMethod method = HttpMethod.GET;

public String httpRequestSendEmail(String action, Object... objs) {
logger.info("httpRequestSendEmail start | action : " + action);
String result = null;
String utcTime = getUTCTimeStr();
String signatureNonce = UUID.randomUUID().toString();
String[] keys = new String[]{"AccessKeyId", "AccountName", "Action", "AddressType", "Format", "HtmlBody",
"RegionId", "ReplyToAddress", "SignatureMethod",
"SignatureNonce", "SignatureVersion", "Subject", "TagName", "Timestamp", "ToAddress", "Version"};
Object[] values = new Object[]{AccessKeyId, AccountName, action, AddressType, Format, HtmlBody, RegionId,
ReplyToAddress, SignatureMethod, signatureNonce
```

```
, SignatureVersion, Subject, TagName, utcTime, ToAddress, Version};

String signature = null;
try {
signature = getSignature(prepareParamStrURLEncoder(keys, values), method);
} catch (Exception e) {
e.printStackTrace();
}
System.out.println("-----Signature-----" + signature);
String[] keys1 = new String[]{"AccessKeyId", "AccountName", "Action", "AddressType", "Format", "HtmlBody",
"RegionId", "ReplyToAddress", "Signature", "SignatureMethod",
"SignatureNonce", "SignatureVersion", "Subject", "TagName", "Timestamp", "ToAddress", "Version"};
Object[] values1 = new Object[]{AccessKeyId, AccountName, action, AddressType, Format, HtmlBody, RegionId,
ReplyToAddress, signature, SignatureMethod, signatureNonce
, SignatureVersion, Subject, TagName, utcTime, ToAddress, Version};

String param = prepareParamStrURLEncoder(keys1, values1);

System.out.println("-----url-----" + protocol + "://" + host + "/?" + param);
if (method == HttpMethod.GET) {
result = doHttpRequest(protocol + "://" + host + "/", param);
} else {
result = doPostHttpRequest(protocol + "://" + host + "/", param);
}

System.out.println("-----httpRequestSendEmail result-----" + result);
return result;
}

/**
* 获取签名
*
* @param param
* @param method
* @return
* @throws Exception
*/
private String getSignature(String param, HttpMethod method) throws Exception {
String toSign = method + "&" + URLEncoder.encode("/", "utf8") + "&"
+ getUtf8Encoder(param);
System.out.println("-----StringToSign-----" + toSign);
byte[] bytes = HmacSHA1Encrypt(toSign, AccessKeySecret + "&");
return Base64.encode(bytes);
}

private String getUtf8Encoder(String param) throws UnsupportedEncodingException {
return URLEncoder.encode(param, "utf8")
.replaceAll("\\+", "%20")
.replaceAll("\\*", "%2A")
.replaceAll("%7E", "~");
}

private static final String MAC_NAME = "HmacSHA1";
private static final String ENCODING = "UTF-8";

/**
```

```
* 使用 HMAC-SHA1 签名方法对对encryptText进行签名
*
* @param encryptText 被签名的字符串
* @param encryptKey 密钥
* @return
* @throws Exception
*/
public static byte[] HmacSHA1Encrypt(String encryptText, String encryptKey) throws Exception {
byte[] data = encryptKey.getBytes(ENCODING);
//根据给定的字节数组构造一个密钥,第二参数指定一个密钥算法的名称
SecretKey secretKey = new SecretKeySpec(data, MAC_NAME);
//生成一个指定 Mac 算法 的 Mac 对象
Mac mac = Mac.getInstance(MAC_NAME);
//用给定密钥初始化 Mac 对象
mac.init(secretKey);

byte[] text = encryptText.getBytes(ENCODING);
//完成 Mac 操作
return mac.doFinal(text);
}

private static DateFormat daetFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

/**
* 得到UTC时间，类型为字符串，格式为"yyyy-MM-dd HH:mm:ss",成功返回"yyyy-MM-ddTHH:mm:ssZ"<br />
* 如果获取失败，返回null
*
* @return
*/
public static String getUTCTimeStr() {
// 1、取得本地时间：
Calendar cal = Calendar.getInstance();
// 2、取得时间偏移量：
int zoneOffset = cal.get(java.util.Calendar.ZONE_OFFSET);
// 3、取得夏令时差：
int dstOffset = cal.get(java.util.Calendar.DST_OFFSET);
// 4、从本地时间里扣除这些差量，即可以取得UTC时间：
cal.add(java.util.Calendar.MILLISECOND, -(zoneOffset + dstOffset));
String date = daetFormat.format(cal.getTime());
System.out.println("时间------" + date);
String[] strs = date.split(" ");
return strs[0] + "T" + strs[1] + "Z";
}

@Test
public void testUTC() {
String UTCTimeStr = getUTCTimeStr();
System.out.println("时间------" + UTCTimeStr);
}

/**
* https://dm.aliyuncs.com/?Action=SingleSendMail
* &AccountName=test@example.com
* &ReplyToAddress=true
* &AddressType=1
* &ToAddress=test1@example.com
```

```
* &Subject=Subject
* &HtmlBody=body
* &<公共请求参数>
* <p>
* POST方式需要修改请求头 Content-Type: application/x-www-form-urlencoded
*/
@Test
public void testSingleSendMail() {
String action = "SingleSendMail";
httpRequestSendEmail(action, "");
}
}
```

温馨提示

```
杭州：
host = "dm.aliyuncs.com" ;
AccountName = "填写杭州地区对应的发信地址";
Version = "2015-11-23";
RegionId = "cn-hangzhou";
新加坡：
host = "dm.ap-southeast-1.aliyuncs.com" ;
AccountName = "填写新加坡地区对应的发信地址";
Version = "2017-06-22";
RegionId = "ap-southeast-1";
澳洲：
host = "dm.ap-southeast-2.aliyuncs.com" ;
AccountName = "填写澳洲地区对应的发信地址";
Version = "2017-06-22";
RegionId = "ap-southeast-2";
```

———-杭州服务区域最后发出的url请求示例如下———-

https://dm.aliyuncs.com/?AccessKeyId=xxxxxxx&AccountName=noreply%40xxxxxxxx.club&Action=SingleSendMail&AddressType=1&Format=JSON&HtmlBody=%3Chtml%3E%3Cbody%3E%3Cimg%20alt%3D%22Go%E8%AF%AD%E8%A8%80%E4%B8%AD%E6%96%87%E7%BD%91%22%20src%3D%22https%3A%2F%2Fstatic.studygolang.com%2Fimg%2Flogo1.png%22%20%3E%3Cimg%20alt%3D%22%22%20src%3D%22https%3A%2F%2Fgoss4.vcg.com%2Fcreative%2Fvcg%2F400%2Fnew%2FVCG211173951082.jpg%22%20%3E%3Ch3%3ETest%20send%20to%20email%20%EF%BC%88%EF%BC%89%EF%BC%81%3C%2Fh3%3E%3C%2Fbody%3E%3C%2Fhtml%3E%20%3Ca%25b%27%20%2B%20%2A%20%257E%3E%20%E6%B5%8B%E8%AF%95%E9%82%AE%E4%BB%B6%E6%AD%A3%E6%96%87%E3%80%82%E4%BD%A0%E6%AD%A4%E6%AC%A1%E7%94%B3%E8%AF%B7%E6%B3%A8%E5%86%8C%E7%9A%84%E9%AA%8C%E8%AF%81%E7%A0%81%E4%B8%BA%EF%BC%9A123456&RegionId=cn-hangzhou&ReplyToAddress=true&Signature=1PysI0HsZeOEHqQ%3D&SignatureMethod=HMAC-SHA1&SignatureNonce=37161234-7747-419f-b433-bc4719504b01&SignatureVersion=1.0&Subject=%E6%B5%8B%E8%AF%95%E4%B8%BB%E9%A2%98&TagName=%E6%B5%8B%E8%AF%95Tag&Timestamp=2018-11-

07T09%3A40%3A13Z&ToAddress=xxxxx%40xxx.com&Version=2015-11-23

发送邮件相关接口

SingleSendMail

描述

SingleSendMail 接口为单一发信接口。

请求参数

名称	类型	是否必须	描述
Action	String	必须	操作接口名，系统规定参数，取值：SingleSendMail。
AccountName	String	必须	管理控制台中配置的发信地址。
ReplyToAddress	Boolean	必须	使用管理控制台中配置的回信地址（状态必须是验证通过）。
AddressType	Number	必须	取值范围 0~1: 0 为随机账号；1 为发信地址。
ToAddress	String	必须	目标地址，多个 email 地址可以用逗号分隔，最多100个地址。
FromAlias	String	可选	发信人昵称,长度小于 15 个字符。 例如:发信人昵称设置为”小红”，发信地址为“test@example.com”，收信人看到的发信地址为"小红"<test@example.com>。
Subject	String	可选	邮件主题，建议填写。

HtmlBody	String	可选	邮件 html 正文，限制 28K。
TextBody	String	可选	邮件 text 正文，限制 28K。
ClickTrace	String	可选	取值范围 0~1: 1 为打开数据跟踪功能; 0 为关闭数据跟踪功能。该参数默认值为 0。

其他请求参数，请参见 **公共请求参数**。

返回参数

公共返回参数，请参见 **公共返回参数**。

错误码

错误代码	描述	Http 状态码	语义
InvalidMailAddress.NotFound	The specified mailAddress does not exist.	400	发信地址不存在。
InvalidMailAddressStatus.Malformed	The specified mailAddress status is wrongly formed.	400	发信地址状态不正确。
InvalidToAddress	The specified toAddress is wrongly formed.	400	目标地址不正确。
InvalidBody	The specified textBody or htmlBody is wrongly formed.	400	邮件正文不正确。textBody 或 htmlBody 不能同时为空。
InvalidSendMail.Spam	Sendmail rejected by spam filter.	400	本次发送操作被反垃圾系统检测为垃圾邮件，禁止发送。请仔细检查邮件内容和域名状态等。
InvalidSubject.Malformed	The specified subject is wrongly formed.	400	邮件主题限制在 100 个字符以内。
InvalidMailAddressDomain.Malformed	The specified mailAddress domain does not exist.	400	发信地址的域名状态不正确，请检查 MX、SPF 配置是否正确。
InvalidFromALias.Malformed	The specified fromAlias is wrongly formed.	400	发信人昵称不正确，请检查发信人昵称是否正确，长度应小于 15 个字符。

InvalidToAddress.Spam	Sendmail rejected by invalid address.	400	收件地址因无效而拒绝，请检查收件地址是否真实有效。

示例

请求示例

```
https://dm.aliyuncs.com/?Action=SingleSendMail
&AccountName=test@example.com
&ReplyToAddress=true
&AddressType=1
&ToAddress=test1@example.com
&Subject=Subject
&HtmlBody=body
&<公共请求参数>
```

返回示例

XML格式

```
<SingleSendMailResponse>
<RequestId>12D086F6-8F31-4658-84C1-006DED011A85</RequestId>
</SingleSendMailResponse>
```

JSON示例

```
{
"RequestId": "12D086F6-8F31-4658-84C1-006DED011A85"
}
```

BatchSendMail

描述

BatchSendMail 接口用于发送批量邮件。

请求参数

名称	类型	是否必须	描述
Action	String	必须	操作接口名，系统规定参数，取值：BatchSendMail。
AccountName	String	必须	管理控制台中配置的发信地址。
AddressType	Number	必须	取值范围 0~1: 0 为随机账号；1 为发信地址。
TemplateName	String	必须	预先创建且通过审核的模板名称。
ReceiversName	String	必须	预先创建且上传了收件人的收件人列表名称。
TagName	String	可选	邮件标签名称
ClickTrace	String	可选	取值范围 0~1: 1 为打开数据跟踪功能; 0 为关闭数据跟踪功能。该参数默认值为 0。

其他请求参数请参见 **公共请求参数**。

返回参数

公共返回参数，详见**公共返回参数**。

错误码

错误代码	描述	Http状态码	语义
InvalidMailAddress.NotFound	The specified mailAddress does not exist.	400	发信地址不存在。
InvalidMailAddressStatus.Malformed	The specified mailAddress status is wrongly formed.	400	发信地址状态不正确。
InvalidMailAddressSendType.Malformed	The specified mailAddress sendType is wrongly formed.	400	发信地址类型不正确，只能使用批量类型的发信地址。
InvalidReceiverName.Malformed	The specified receiver name is wrongly formed.	400	收件人列表名称不正确。列表不存在或者列表为空。

InvalidSendMail.Spam	Sendmail rejected by spam filter.	400	本次发送操作被反垃圾系统检测为垃圾邮件，禁止发送。请检查并优化邮件内容。
InvalidTemplateName.Malformed	The specified template name is wrongly formed.	400	模板名称不正确。
InvalidMailAddressDomain.Malformed	The specified mailAddress domain does not exist.	400	发信地址的域名状态不正确。请检查域名状态，所有权、MX、SPF配置是否正确。

示例

请求示例

```
https://dm.aliyuncs.com/?Action=BatchSendMail
&AccountName=test@example.com
&AddressType=1
&TemplateName=test1
&ReceiversName=test2
&TagName=test3
&<公共请求参数>
```

返回示例

XML格式

```
<BatchSendMailResponse>
<RequestId>12D086F6-8F31-4658-84C1-006DED011A85</RequestId>
</BatchSendMailResponse>
```

JSON示例

```
{
"RequestId": "12D086F6-8F31-4658-84C1-006DED011A85"
}
```

错误代码表

客户端错误

详见各个接口的错误码:

- SingleSendMail。
- BatchSendMail。

服务器端错误

错误代码	描述	Http 状态码	语义
ServiceUnavailable	The request has failed due to a temporary failure of the server.	503	服务不可用。
InternalError	The request processing has failed due to some unknown error, exception or failure.	500	内部错误。
MissingParameter	The input parameter <parameter name> that is mandatory for processing this request is not supplied	400	缺少必填参数。
Forbidden	User not authorized to operate on the specified resource.	400	没有权限使用该操作，请检查 RAM 的子账户是否赋予邮件推送服务的权限。