sdk-mq-client消息客户端开发使用手册

目 录

[1 概述 1](#_Toc455161988)

[2 开发集成 2](#_Toc455161989)

[2.1 Maven集成 2](#_Toc455161990)

[2.2 配置 2](#_Toc455161991)

[3 消息对象说明 4](#_Toc455161992)

[4 消息发送代码开发 5](#_Toc455161993)

[5 消息接受开发开发 6](#_Toc455161994)

[5.1 发布订阅消息接受 6](#_Toc455161995)

[5.2 队列消息接受 6](#_Toc455161996)

[6 消息集成开发Demo 7](#_Toc455161997)

[7 基于Rest接口的消息发送和接受 10](#_Toc455161998)

[7.1 基于Rest接口的消息发送 10](#_Toc455161999)

[7.2 基于Rest接口的消息接受 11](#_Toc455162000)

[7.3 基于Rest接口的消息发送和接受Demo 14](#_Toc455162001)

[8 图表目录 15](#_Toc455162002)

# 概述

sdk-mq-client 是消息集成客户端，实现了消息的发送和接受功能，便于各业务系统能方便的使用消息服务。

sdk-mq-client当前默认使用的消息服务器是kafka(0.8.2版本)，后续可以扩展集成其他不同的消息服务器类型。

sdk-mq-client使用的消息服务器kafka由统一认证与运营平台提供。

# 开发集成

sdk-mq-client是基于Maven开发的，所以，如果业务系统也是基于Maven开发的，可以通过下面的Maven集成来下载sdk-mq-client以及相关包

## Maven集成

Maven私服：<http://192.168.15.208:8081/nexus/content/groups/public/>

Maven坐标：

<dependency>

<groupId>com.hisign</groupId>

<artifactId>sdk-mq-client</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

## 配置

sdk-mq-client需要如下几个配置参数：

1. 配置参数说明

|  |  |
| --- | --- |
| **配置项** | **描述** |
| zkconnect | Kafka依赖的zookeeper地址列表，例如:  对于单个kafka节点，地址如下:  172.16.0.113:52181  对于kafka集群，地址如下:  172.16.0.113:52181, 172.16.0.114:52181, 172.16.0.115:52181 |
| msgservertype | 消息服务器类型，默认kafka |
| kafka\_brokerlist | Kafka的broker地址列表，例如:  对于单个kafka节点，地址如下:  172.16.0.113: 59092  对于kafka集群，地址如下:  172.16.0.113: 59092, 172.16.0.114: 59092, 172.16.0.115: 59092 |
| kafka\_zkroot | Kafka在zookeeper上的根节点定义，默认kafka |

说明： 上面这些参数配置默认是在统一认证和运营平台中配置的，sdk-mq-client会

自动读取，使用时不需要单独去进行配置，也就是业务系统代码不用管这些

参数的配置。

# 消息对象说明

1. 消息对象属性

|  |  |  |
| --- | --- | --- |
| **属性** | **类型** | **描述** |
| messageNo | long | 消息序列号，自动生成 |
| src | String | 消息源地址(例如发送消息ip地址),自动获取 |
| topic | String | 消息Topic名 |
| type | long | 消息类型，不同系统消息使用不同范围 |
| userObject | Object | 消息实体，可以是String，Integer等普通类型，也可以是集合对象List,Map等，也可以是用户自定义对象，但自定义对象必须实现Serializable接口 |
| additional | String | 附加信息 |
| isCompress | boolean | 是否压缩true:压缩 false不压缩，默认false |

**说明：**

消息对象对应的类是**com.hisign.sdk.msg.Message**

用户定义一个消息对象，需要设置的是topic，type，userObject这三个参数

**topic** 定义了消息发送给那个topic

1. 当前kafaka中的topic是自动创建的，不用事先定义。
2. 各业务系统topic名称以{SystemID}\_{TopicName}格式组织，

其中SystemID是业务系统唯一英文标志，TopicName是topic名称

在topic名前面加上业务系统标志是为了防止不同业务系统之间的

topic名称重复

**type** 对于同一个topic的消息，又可以按照type进行区分不通的消息类型

**userObject** 实际的消息传输实体，用户要发送的消息内容是封装在这里的，该消息

实体可以是String，Integer等普通类型，也可以是集合对象List,Map等，

也可以是用户自定义对象，但自定义对象必须实现Serializable接口，

并且需要显式实现空参构造函数。(因为默认使用kryo序列化要求)

**消息Message建立参照下面代码**

Message msg = **new** Message();

msg .setType(600001); //新增用户

msg.setUserObject(**person**);

msg.setTopic("UAOP\_test");

其中：

person是一个用户自定义消息对象，

600001是消息类型，代表是新增用户

UAOP\_test是topic名称，其中UAOP是业务系统唯一标识

# 消息发送代码开发

方法1：

MessageClient.getInstance().send(message);

方法2：

MessageClient.getInstance().send(“UAOP\_test”，message);

其中方法1，是在message中已经指定了Topic，所以可以调用send发送，方法2是

按照send方法中指定的topic进行消息发送

# 消息接受开发开发

## 发布订阅消息接受

MsgClient.getInstance().addMessageHandler(String topicName, MessageHandler handler)

其中

topicName 是消息topic名称

MessageHandler 是消息处理接口类，包含一个消息处理方法

onMessage(Message msg);

## 队列消息接受

MsgClient.getInstance().addQueueMessageHandler(String queueName, MessageHandler handler)

其中

queueName 是消息topic名称

MessageHandler 同上发布订阅模式中的消息处理类

# 消息集成开发Demo

1. Person对象定义

**import** java.io.Serializable;

**public** **class** Person **implements** Serializable {

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **long** id;

**private** String name;

**private** **int** sex;

**public** Person() {

}

**public** Person(**long** id, String name, **int** sex) {

**this**.id = id;

**this**.name = name;

**this**.sex = sex;

}

**public** **long** getId() {

**return** id;

}

**public** **void** setId(**long** id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getSex() {

**return** sex;

}

**public** **void** setSex(**int** sex) {

**this**.sex = sex;

}

@Override

**public** String toString() {

StringBuilder builder = **new** StringBuilder();

builder.append("Person [id=");

builder.append(id);

builder.append(", name=");

builder.append(name);

builder.append(", sex=");

builder.append(sex);

builder.append("]");

**return** builder.toString();

}

1. 消息发送

**import** com.hisign.sdk.msg.Message;

**import** com.hisign.sdk.msg.MessageClient;

**public** **class** MsgSendDemo {

**public** **void** sendMessage(){

//构建消息

Person p1 = **new** Person(1,"张三",1);

Message msg1 = **new** Message();

msg1.setType(600001); //新建Person

msg1.setTopic("YWXT\_topic1"); //消息topic

msg1.setUserObject(p1);

**try** {

MessageClient.*getInstance*().send(msg1);

System.***out***.println("send msg1 successfully!");

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** **void** main(String[] args) {

MsgSendDemo demo = **new** MsgSendDemo();

demo.sendMessage();

}

}

1. 消息接受

**import** com.hisign.sdk.msg.Message;

**import** com.hisign.sdk.msg.MessageClient;

**import** com.hisign.sdk.msg.MessageHandler;

**public** **class** MessageRecieveDemo {

**public** **void** receiveMsg(){

**try**{

MessageClient.*getInstance*().addMessageHandler("YWXT\_topic1", **new** MessageHandler(){

@Override

**public** **void** onMessage(Message msg) {

**if**(msg.getType() == 600001){ //只处理新建Person消息类型

Person p1 = (Person)msg.getUserObject();

System.***out***.println("receive msg userObject="+p1.toString());

}

}

});

System.***out***.println("start receiveMsg completey!");

}**catch**(Exception ex){

ex.printStackTrace();

}

}

**public** **static** **void** main(String[] args) **throws** Exception {

System.*setProperty*("SYSTEMID", "YWXT");

MessageRecieveDemo demo = **new** MessageRecieveDemo();

demo.receiveMsg();

System.***in***.read(); //等待，防止直接进程退出

}

}

注意：业务系统在使用消息接受时，需要使用

System.*setProperty*("SYSTEMID", "XXX");

设置下系统唯一标识，因为默认消息分租会使用它作为分租标识的一部分，如下：

defaultGroup + "\_" + systemID+"\_"+ip

# 基于Rest接口的消息发送和接受

基于kafka-rest，可以基于Rest接口进行kafka消息的发送和接受，sdk-mq-client在序列化时，采用了JSON进行消息序列化格式，因此，基于sdk-mq-client实现的Java业务系统应用可以与非Java(比如C#,Python,Javascript)等任意语言进行消息的发送和接受通信，也就是Java语言开发的业务系统可以发送消息到非Java语言的rest应用上，Java语言开发的业务系统也可以接受任意非Java语言的rest接口发送的消息。

下面以curl操作来展示rest消息发送和接受。

## 基于Rest接口的消息发送

**发送消息请求**

curl -X POST -H "Content-Type: application/vnd.kafka.json.v1+json" \

--data '{"records":

[{"value":

{"addtional":"",

"compress":false,

"messageNo":1,

"src":"192.168.40.24",

"topic":"YWXT\_topic1",

"type":600001,

"**userObject**":

{"id":1,"name":"张三","sex":1},

"**userObjectClass**":

"com.hisign.sdk.msg.demo.Person"}

}]

}' \

"http://**172.16.0.114**:8082/topics/**YWXT\_topic1**"

**正确结果返回**

{"offsets":[{"partition":0,"offset":6,"error\_code":null,"error":null}],"key\_schema\_id":null,"value\_schema\_id":null}

**说明：**

1. 消息请求格式严格按照上面的格式。
2. 黄色区域是我们的消息体，具体参数说明请参照第三节”消息对象说明”
3. userObject定义的是真实的消息实体，如果要与Java应用进行消息通信，需要在

userObjectClass指定对应的Java消息实体类型(可以是自定义类型和Java自身类型)，

如果没有指定userObjectClass，则Java应用中默认把userObject转换成HashMap。

1. http://**172.16.0.114**:8082/topics/**YWXT\_topic1**中

**172.16.0.114**  对应kafka服务器IP

**YWXT\_topic1** 对应Topic名称

## 基于Rest接口的消息接受

给予Rest消息接受，需要发送三个请求步骤，分别是创建Consumer实例，从Topic查询消息，关闭Comsumer，明令分别如下：

**步骤1-创建Consumer实例：**

curl -X POST -H "Content-Type: application/vnd.kafka.v1+json" \

--data '{"name": "my\_consumer\_instance", "format": "json", "auto.offset.reset": "smallest"}, "auto.commit.enable": "true"}' \

http://172.16.0.114:8082/consumers/my\_json\_consumer

正确结果返回

{"instance\_id":"my\_consumer\_instance","base\_uri":"**http://172.16.0.114:8082/consumers/my\_json\_consumer/instances/my\_consumer\_instance**"}

注意红色部分结果会在步骤2和步骤3中使用到。

**步骤2-接受消息：**

curl -X GET -H "Accept: application/vnd.kafka.json.v1+json" \

http://172.16.0.114:8082/consumers/my\_json\_consumer/instances/my\_consumer\_instance/topics/YWXT\_topic1

说明: 黄色背景是步骤1返回的结果，YWXT\_topic1是topic名称

返回结果，如果有消息数据，则返回消息数据json串，如下：

[{"key":null,

"value":

{"addtional":"",

"compress":false,

"messageNo":1,

"src":"192.168.40.24",

"topic":"YWXT\_topic1",

"type":600001,

"userObject":

{"id":1,"name":"张三","sex":1},

"userObjectClass":

"com.hisign.sdk.msg.demo.Person"

},

"partition":0,

"offset":25

}]

如果没有消息数据，则返回空

**步骤3-关闭Consumer实例：**

curl -X DELETE \

http://172.16.0.114:8082/consumers/my\_json\_consumer/instances/my\_consumer\_instance

返回结果：

空

其实返回{"error\_code":40403,"message":"Consumer instance not found."}也不用管。

## 基于Rest接口的消息发送和接受Demo

下面提供基于HttpClient实现的Rest消息发送和接受Demo



# 图表目录

**附图目录：**

**未找到图形项目表。**

**附表目录：**

[表格1. 配置参数说明 2](#_Toc455161978)

[表格2. 消息对象属性 4](#_Toc455161979)