SOFAStack

消息队列 DMS

使用指南

产品版本：AntStack Plus 1.11.0

文档版本：20220930

蚂蚁集团
ANT GROUP

法律声明

蚂蚁集团版权所有©2022，并保留一切权利。

商标声明

免责声明

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

目录

1.概述

消息队列（Message Queue）产品，作为一种典型的消息代理组件（Message Broker），是企业级应用系统中常用的消息中间件，主要应用于分布式系统或组件之间的消息通讯，提供具有可靠、异步和事务等特性的消息通信服务。应用消息代理组件可以降低系统间耦合度，提高系统的吞吐量、可扩展性和可用性。

消息队列产品主要涉及五个核心角色，消息发布者（Publisher）、消息代理组件（Message Broker）、消息订阅者（Subscriber）、消息类型（Message Type）和订阅关系（Binding），具体描述如下：

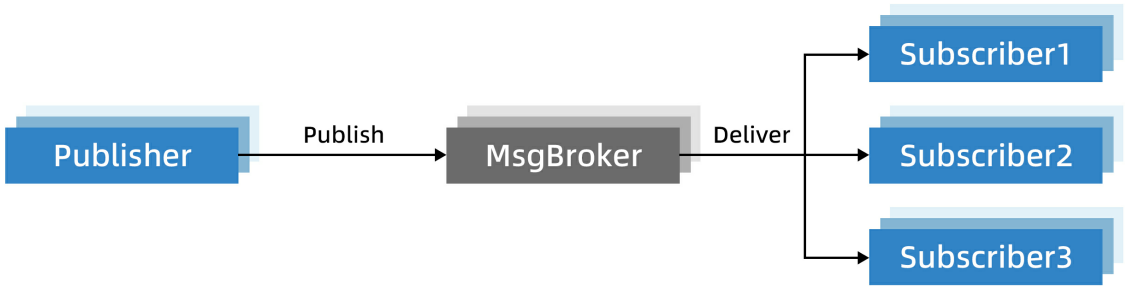
- **消息发布者**：指发送消息的应用系统。一个应用系统可以发送一种或者多种类型的消息，消息发布者将消息发送到消息代理组件。
- **消息代理组件**：负责接收发布者发送的消息，根据消息类型和消息订阅元数据将消息分发投递到一个或多个消息订阅者。整个过程涉及消息类型校验、消息持久化存储、消息订阅关系匹配、消息投递和消息恢复等核心功能。
- **消息订阅者**：指订阅消息的应用系统。一个应用系统可以订阅一种或者多种消息类型，消息订阅者收到的消息来自消息代理组件。
- **消息类型**：一种消息类型由 TOPIC 和 EVENT CODE 唯一标识。
- **消息订阅**：用来描述一种消息类型被哪些订阅者订阅，订阅元数据也被称为 Binding。

产品优势

- 可为不同应用系统间提供可靠的消息通信，降低系统间耦合度并提高整体架构的可扩展性和可用性。
- 可为不同应用系统间提供异步消息通信，提高系统吞吐量和性能。
- 发布者系统、消息代理组件以及订阅者系统均支持集群水平扩展，可依据业务消息量动态部署计算节点。
- 支持事务型消息，保证消息与本地数据库事务的一致性。

实现原理

消息发布者 与 **消息订阅者** 是一对多的关系，即一个消息类型由一个消息发布者发出，但可以被一个或者多个订阅者接收。



注意事项

- 消息订阅者收到的消息不保证有序，即收到消息的顺序与发布者发送消息的顺序可能会不一致。
- 消息投递策略为至少一次（At-least-once），即对于同一条消息消息订阅者可能收到多次，要求订阅者保证消息处理幂等特性。

2.快速开始

使用消息队列产品实现应用系统间发布和订阅异步消息的步骤为：

1. 配置 SOFABoot 工程
2. 确定消息类型
3. 本地开发
    i. 发布者（Publisher）
    ii. 消费者（Consumer）
4. 在控制台配置元数据
5. 发布应用

配置 SOFABoot 工程

消息队列工程的开发基于 SOFABoot 框架。您必须已经创建一个 SOFABoot 工程，并完成蚂蚁中间件 Maven 依赖的必要配置。 详情参见：

- SOFABOOT 环境搭建要求
- SOFABOOT 快速入门

确定消息类型

异步消息通信涉及两个核心角色：**消息发布者** 和 **消息消费者**。同一个应用系统可能兼具消息发布者和消息消费者两个角色。发送消息的应用系统需要配置发布者，接收消息的应用系统配置消费者。两者都必须遵循事先约定好的消息类型。

一种消息类型由 TOPIC 和 EVENT CODE 唯一标识。在本地开发前，我们需要首先确定这两个参数。确定好消息类型之后，消息发布者（Publisher）与消息消费者（Consumer）都必须遵循它。

参数	命名规则	示例
TOPIC	以 “TP_” 开头	TP_DEFAULT
EVENT CODE	以 “EC_” 开头	EC_DEFAULT

本地开发

。

消息队列项目示例代码位于 `middleware-v2/mqtutorial-all-in-one` 文件夹下。

无论您将应用作为消息发布者还是消费者，都必须在工程中引入消息队列 Maven 依赖。在异步消息模块的 `pom.xml` 文件中添加如下依赖：

```
<dependency>
    <groupId>com.alipay.sofa</groupId>
    <artifactId>mq-enterprise-sofa-boot-starter</artifactId>
</dependency>
```

示例工程代码默认使用云上部署配置。如果您需要进行本地调试，请修改 `application.properties` 文件中的 `run.mode` 参数，设置为 DEV 模式：

```
# 文件路径： middleware-v2/mqtutorial-all-in-one/app/web/src/main/resources/config/application.properties
run.mode=DEV
```

发布者（Publisher）

发布者配置

发送消息的应用系统需要在 Spring 应用上下文文件中配置 `sofa:publisher` 。应用上下文文件的默认路径如下所示：

```
src/main/resources/META-INF/xxx/xxx-xxx.xml
```

发布者配置示例如下：

```
<sofa:publisher id="mqPublisher" group="P_appname_service">
      <sofa:channels>
            <sofa:channel value="TP_DEFAULT"/>
      </sofa:channels>
      <sofa:binding.msg_broker/>
</sofa:publisher>

<bean id="mqService" class="com.antcloud.tutorial.mq.endpoint.service.MqService">
    <property name="mqPublisher" ref="mqPublisher"/>
</bean>
```

- **id**：Spring Bean 的单例服务标识，可以被依赖注入到其它 Spring Bean 属性值中。
- **group**：命名格式为 `P\_应用名\_服务名` ，是发布者唯一标识，不允许存在两个 `sofa:publisher` 配置相同的 group 属性值。
- **sofa:channel**：sofa:channel 元素的 **value** 值是此发布者发送的消息类型 TOPIC 值，如果发送消息类型涉及多个 TOPIC，必须配置多个 `sofa:channel` 元素。

发布者服务实现

消息发布者服务实现包含三个步骤：

1. 创建消息对象（UniformEvent）。
2. 设置消息对象属性值。
3. 发送消息。

消息对象由 `UniformEventBuilder` 负责创建，必须指定 TOPIC 和 EVENTCODE 两个参数。消息负载封装在业务对象中，并设置为 uniformEvent 的 payload 属性值。消息发送通过 `uniformEventPublisher` 对象的 `publisherUniformEvent` 方法完成，此方法可能抛出运行时异常，如果应用程序预期捕获异常并处理，必须设置 uniformEvent 对象的属性值 `throwExceptionOnFailed` 为 true，否则运行时异常不会被抛出，只会记录错误日志。

> **说明**
>
> `UniformEvent.throwExceptionOnFailed` 字段的默认值是 false。

消息发布者实现代码示例如下所示：

```
public class MqService {

    private static final Logger   logger              = LoggerFactory.getLogger(MqService.c
lass);

    private final static String   TOPIC               = "TP_DEFAULT";

    private final static String   EVENTCODE           = "EC_DEFAULT";

    private UniformEventPublisher mqPublisher;

    private UniformEventBuilder   uniformEventBuilder = new DefaultUniformEventBuilder();

    public boolean publish() {
        if (logger.isInfoEnabled()) {
            logger.info("Publish a message.");
        }

        /* Create a message instance. */
        final UniformEvent message = uniformEventBuilder.buildUniformEvent(TOPIC, EVENTCODE
);

        /* Set the business object as an event payload. */
        message.setEventPayload(buildDefaultAccount());

        /* Mark that a runtime exception must be thrown when publishing failure. */
        message.setThrowExceptionOnFailed(true);

        boolean publishSuccess = false;

        try {
            /* Do publish action. */
            mqPublisher.publishUniformEvent(message);
            publishSuccess = true;

            logger.info("Public a message, success. TOPIC [{}] EVENTCODE [{}] id [{}] paylo
ad [{}]", new Object[] {
                    message.getTopic(), message.getEventCode(), message.getId(), message.ge
tEventPayload() });
```

```
        } catch (Exception e) {
            logger.error("Public a message, failure. TOPIC [{}] EVENTCODE [{}] error [{}]",
                new Object[] { message.getTopic(), message.getEventCode(), e.getMessage() }
, e);
        }

        return publishSuccess;
    }

    private Account buildDefaultAccount() {
        Account account = new Account();
        account.setId(UUID.randomUUID().toString());
        account.setAmount(new Random().nextDouble());
        account.setGmtCreate(new Date());
        return account;
    }

    public void setMqPublisher(UniformEventPublisher mqPublisher) {
        this.mqPublisher = mqPublisher;
    }
}
```

消费者（Consumer）

消费者配置

> **说明**
>
> 消息消费者也可叫做消息订阅者（Subscriber）。

订阅消息的应用系统需要在 Spring 应用上下文文件中配置 `sofa:consumer` 。应用上下文文件的默认路径如下所示：

```
src/main/resources/META-INF/xxx/xxx-xxx.xml
```

消息消费者配置示例：

```
<!-- Declare a consumer bean with id "mqConsumer" -->
<sofa:consumer id="mqConsumer" group="S_appname_service">
    <sofa:listener ref="mqMessageListener"/>
        <sofa:channels>
        <sofa:channel value="TP_DEFAULT">
            <sofa:event eventType="direct" eventCode="EC_DEFAULT" persistence="true"/>
         </sofa:channel>
    </sofa:channels>
    <sofa:binding.msg_broker/>
</sofa:consumer>

<!-- mq message listener -->
<bean id="mqMessageListener" class="com.antcloud.tutorial.mq.endpoint.service.MqMessageList
ener"/>
```

- **id**：pring Bean 的单例服务标识。
- **group**：命名格式为 `S\_应用名\_服务名`，是消费者唯一标识，不允许存在两个 `sofa:consumer` 配置相同的 group 属性值。
- **sofa:listener**：sofa:listener 元素的 ref 属性值设置为消息接监听器单例服务标识。
- **sofa:channel**：sofa:channel 元素的 value 值是此消费者订阅的消息类型 TOPIC 值，如果订阅的消息类型涉及多个 TOPIC，必须配置多个 `sofa:channel` 元素。
- **sofa:event**：sofa:event 元素配置具体的消息订阅信息，eventType 属性值默认设置为 `direct`，eventCode 属性值设置为消息类型 eventcode 值，persistence 属性值设置为持久订阅（true）或者非持久订阅（false）。

消息接收监听器实现

订阅消息的应用系统必须实现 `com.alipay.common.event.UniformEventMessageListener` 接口并配置在 `sofa:listener` 元素的 ref 属性值中。当消息被消息消费者接收到时，`com.alipay.common.event.UniformEventMessageListene.onUniformEvent` 会被调用，应用系统通过实现此方法执行消息消费逻辑。消息接收监听器实现如下所示：

```
public class MqMessageListener implements UniformEventMessageListener {

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

    @Override
    public void onUniformEvent(UniformEvent message, UniformEventContext context) throws Ex
ception {

        /* get TOPIC, EVENTCODE and payload from the message instance */
        final String topic = message.getTopic();
        final String eventcode = message.getEventCode();
        final String id = message.getId();
        final Object businessObject = message.getEventPayload();

        logger.info("Receive a message, TOPIC [{}] EVENTCODE [{}] id [{}] payload [{}]", ne
w Object[] { topic,
                eventcode, id, businessObject });

        try {
            boolean processSuccess = processMessage(businessObject);

            if (!processSuccess) {
                /* Process the message failure, set cause and rollback, the message is re-d
elivered later. */
                context.setContextDesc("process error cause");
                context.setRollbackOnly();
            }
        } catch (Exception e) {
            logger.error("Process a message, failure. TOPIC [{}] EVENTCODE [{}] id [{}] err
or {}", new Object[] {
                    topic, eventcode, id, e.getMessage() }, e);
            /* When any exception is thrown, the message is re-delivered later. */
            throw e;
        }
    }

    /* Process the business logic */
    private boolean processMessage(Object businessObject) {
        return true;
    }
}
```

消费者应用系统接收到消息后可能存在以下三种处理策略：

- 消息消费正常。
- 消息无法消费，主动回滚并设置回滚原因。消息会被重新投递。
- 消息消费异常，抛出未捕获异常。消息会被重新投递。

在控制台配置元数据

消息类型和消息订阅两个必须的元数据信息必须在消息队列控制台中配置元数据后才能正常使用。管控台中配置的元数据信息必须和本地代码中一致。

消息类型元数据配置

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息类型管理**。
3. 单击 **新增消息类型**，在 **新增消息类型** 面板，配置以下信息：

   - **消息主题**：本例中输入 `TP_DEFAULT` 。
   - **消息事件码**：本例中输入 `EC_DEFAULT` 。
   - **描述**：本例中输入 `默认消息类型` 。
4. 单击 **提交**。

消息订阅元数据配置

1. 在左侧导航栏上，单击 **消息订阅管理**。
2. 单击 **添加消息订阅**，在 **添加消息订阅** 面板，配置以下信息：

消息队列支持两种消息订阅类型：**DIRECT** 以及 **HEADER**。DIRECT 支持消息订阅者接收所有消息，HEADER 支持消息订阅者对消息进行过滤设置。您可根据实际情况选择消息订阅类型。详见 消息订阅类型。

3. 单击 **提交**。

发布应用

SOFABoot 应用的发布指南参见文档 SOFABOOT 云端发布。

3.操作指南

3.1. 消息类型管理

3.1.1. 消息类型概述

消息类型由 TOPIC（消息主题）和 EVENTCODE（消息事件码）进行唯一标识。本地应用开发中，消息发布者（Publisher）与消息消费者（Consumer）都必须遵循该消息类型。同时，您必须要在消息队列控制台配置消息类型数据，消息队列才能接收对应的消息类型。

消息类型列表

在 **消息队列控制台 > 消息类型管理** 页面中，以列表形式展示了现有的消息类型，包括以下信息：

- **消息主题**：消息的主题，以“TP_”开头。
- **消息事件码**：消息的事件码，以“EC_”开头。
- **描述**：消息类型自定义的描述信息。
- **创建时间**：该消息类型的具体创建时间。
- **操作**：支持查看消息类型详情，修改或删除消息类型，详见 管理消息类型。

如当前列表的消息类型过多，您可以通过列表上放的搜索框，筛选 **消息主题** 和 **消息事件码**，快速查询到想要的消息类型。

消息类型详情

在消息类型列表页，选择想要查看的消息类型，点击 **操作** 列下的 **详情**，即可查看消息类型的详细信息，包括 **消息主题**、**消息事件码**、**描述**、**创建时间**、**修改时间** 和 **操作人**。

3.1.2. 配置消息类型

本文介绍如何在消息队列控制台新增、修改以及删除一个消息类型。

新增消息类型

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息类型管理**。
3. 单击 **新增消息类型**，在 **新增消息类型** 面板，配置以下信息：

参数	是否必填	说明
消息主题	必填	大写字母、数字、下划线，以 “TP_” 开头。

参数	是否必填	说明
消息事件码	必填	大写字母、数字、下划线，以 “EC_” 开头。
描述	选填	建议根据业务意义自定义描述，方便管理维护。

4. 单击 **提交**。

修改消息类型

1. 在消息类型列表页，找到想要修改的消息类型，单击对应 **操作** 列下的 **修改**。
2. 在 **修改消息类型** 面板，修改 **消息事件码** 或 **描述**。

3. 单击 **提交**。

删除消息类型

1. 在消息类型列表页，找到想要修改的消息类型，单击对应 **操作** 列下的 **删除**。
2. 在弹出的提示对话框中，单击 **确定**。

3.1.3. 导出导入消息类型

在消息队列控制台，您可以批量导入导出消息类型数据，满足消息类型数据备份和迁移的需求。

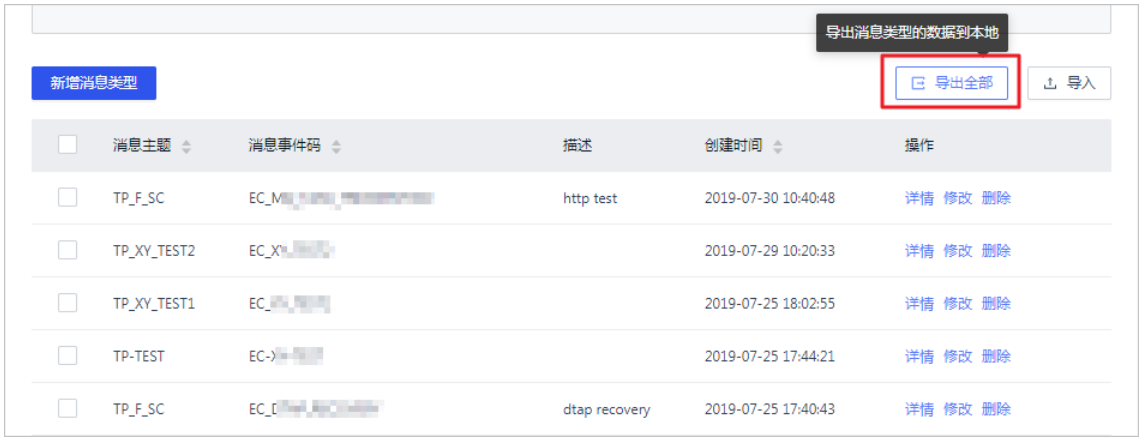
- 导出消息类型：将现有的消息类型数据保存为本地 CSV 文件。
- 导入消息类型：将 CSV 格式的消息类型数据批量导入至消息队列。

导出消息类型

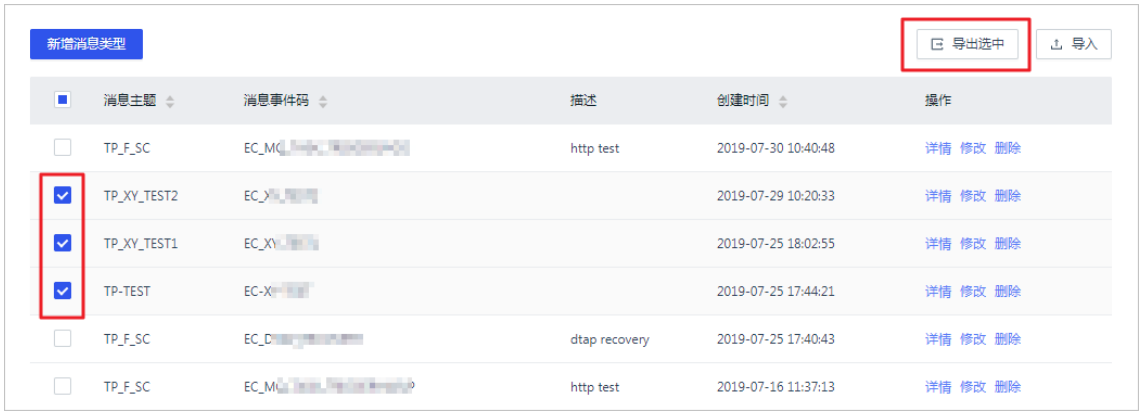
1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息类型管理**。

   消息类型导出支持 **导出全部** 以及 **导出选中** 两种模式：

   - 直接单击列表右上方的 **导出全部**，可将所有消息类型数据以 CSV 格式保存至本地。



- 在列表中，选择部分消息类型，单击 **导出选中**，可将选中的消息类型数据保存至本地。



导入消息类型

1. 在消息类型列表页，单击 **导入**。
2. 在 **导入消息类型** 面板，单击 **上传文件** 并选择本地 CSV 文件。

> 说明
> - 文件必须为 CSV 格式，内容包括消息主题、消息事件码及描述等。
> - 支持下载导入文件模板。

3. 单击 **确定导入**。

3.2. 消息订阅管理

3.2.1. 消息订阅概述

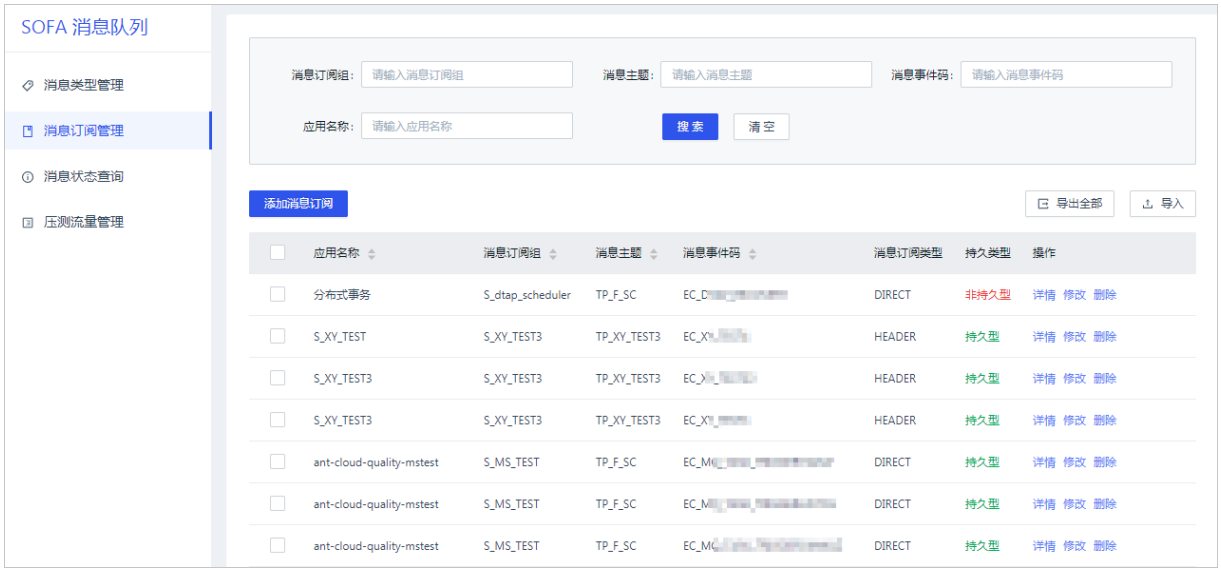
消息订阅用于描述订阅者（也称为消费者，即订阅消息的应用系统）订阅指定消息类型，包括订阅者唯一标识、消息类型、订阅关系类型和持久类型四个元素。只有在消息队列控制台配置消息订阅元数据后，消息队列才会按照消息类型维度将消息投递到指定订阅者。

消息订阅列表

在 **消息队列控制台** > **消息订阅管理** 页面中，以列表形式展示了现有的消息订阅，提供了以下信息：

- **应用名称**：订阅消息的应用系统名称。

- **消息订阅组**：即订阅者唯一标识 Group，支持英文字母、数字、下划线。命名约定格式是 `S\_appname\_service`，其中“S_”前缀代表 Subscriber。
- **消息主题**：要订阅消息的消息主题，与消息事件码一起标识消息类型。
- **消息事件码**：要订阅消息的消息事件码，与消息主题一起标识消息类型。
- **消息订阅类型**：
  - **DIRECT**：消息订阅者会收到指定消息类型的全部消息。
  - **HEADER**：通过设置自定义过滤表达式对消息进行过滤，以实现只接收符合业务数据条件的部分消息。详见 HEADER 消息订阅类型。
- **持久类型**：**持久型** 或 **非持久型**。
- **操作**：支持查看消息订阅详情，修改或删除消息订阅，详见 管理消息订阅。



如当前列表的消息订阅过多，您可以通过列表上方的搜索框，筛选快速查询到想要的消息订阅数据。

消息订阅详情

在消息订阅列表页面，选择想要查看的消息订阅，点击 **操作** 列下的 **详情**，即可查看消息类型的详细信息，包括 **应用名称**、**消息订阅组**、**消息主题**、**消息事件码**、**持久类型**、**消息订阅类型**、**创建时间**、**修改时间** 以及 **操作人**。

3.2.2. 配置消息订阅

本文介绍如何在消息队列控制台添加、修改以及删除一条消息订阅元数据。

新增消息订阅

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息订阅管理**。
3. 单击 **添加消息订阅**，在 **添加消息订阅** 面板，配置以下信息：

- **应用名称**：订阅消息的应用系统名称。
- **消息订阅组**：即订阅者唯一标识 Group，支持英文字母、数字、下划线。命名约定格式是 `S\_appname\_service`，其中“S_”前缀代表 Subscriber。
- **消息主题**：要订阅消息的消息主题，与消息事件码一起标识消息类型。
- **消息事件码**：要订阅消息的消息事件码，与消息主题一起标识消息类型。
- **持久类型**：包括 **持久型** 或 **非持久型**。
- **消息订阅类型**：
  - **DIRECT**：消息订阅者会收到指定消息类型的全部消息。
  - **HEADER**：通过设置自定义过滤表达式对消息进行过滤。详见 HEADER 消息订阅类型。

4. 单击 **提交**。

修改消息订阅

1. 在消息订阅列表中，找到想要修改的消息订阅，单击对应 **操作** 列下的 **修改**。
2. 在 **修改消息订阅** 面板中，支持修改 **消息事件码**、**持久类型**、**消息订阅类型** 等信息。

3. 单击 **提交**。

删除消息订阅

1. 在消息订阅列表中，找到想要修改的消息订阅，单击对应 **操作** 列下的 **删除**。
2. 在弹出的确认窗口中，单击 **确定**。

3.2.3. 导出导入消息订阅

在消息队列控制台，您可以批量导入导出消息订阅的数据。

- 导出消息订阅：将现有的消息订阅数据保存为本地 CSV 文件。
- 导入消息订阅：将 CSV 格式的消息订阅数据批量导入至消息队列。

导出消息订阅

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息订阅管理**。

   消息订阅导出支持 **导出全部** 以及 **导出选中** 两种模式：

   - 直接点击列表右上方的 **导出全部**，即可将所有消息订阅数据以 CSV 格式保存至本地。

- 在列表中，选择部分消息订阅，点击 **导出选中** 按钮，可将选中的消息订阅数据保存至本地。

导入消息订阅

1. 在消息订阅列表页，单击 **导入**。
2. 在 **导入消息订阅关系** 面板，单击 **上传文件** 并选择本地 CSV 文件。

> 说明
> - 文件必须为 CSV 格式，内容包括应用名称、订阅组、消息主题及消息事件码等。
> - 支持下载导入文件模板。

3. 单击 **确定导入**。

3.3. 查询消息状态

在消息队列控制台，您可以查询指定消息的消息状态，也可以对积压消息按消息主题进行查询，并根据业务需求删除无用的消息对象。

按消息 ID 查询

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息状态查询**。
3. 选择 **按消息 ID 查询**。

4. 在页面上方的 **搜索消息 ID** 栏中，输入完整的消息 ID。
5. 单击  ，即可获取指定消息的消息状态，包括以下信息：
   - 基本信息：
     - **消息 ID**
     - **消息主题**
     - **消息事件码**
     - **消息事务状态**：包括 **已提交**、**未提交**。
     - **消息投递状态**：包括 **已投递**、**未投递**。
   - 时间信息：包括 **消息创建时间**、**消息上次投递时间**、**消息下次投递时间**。
   - 投递信息：包括 **消息发布者分组**、**消息已投递次数**、**已投递消息订阅者分组**、**未投递消息订阅者分组**。

按消息主题查询

1. 在左侧导航栏上，单击 **消息状态查询**。
2. 选择 **按消息主题查询**。
3. 在页面上方的搜索框中，输入或选择以下信息：
   - **消息主题**、**消息事件码**：必填，用以标识消息类型。
   - **时间范围**：支持查询最近 15 天的消息。
4. 单击 **搜索**，即可获取指定范围内的消息列表，列表提供以下信息：
   - **消息 ID**：消息对象的唯一标识。
   - **消息事件码**：该消息对象所属的消息类型的 EVENT CODE。
   - **消息状态**：包括 **已投递**、**未投递**。
   - **未投递消息订阅者分组**：未投递的消息订阅者分组。
   - **创建时间**：消息的创建时间。
   - 操作：
     - 点击 **详情**，可查看该消息的具体信息。
     - 点击 **删除**，确认后可删除该消息。

批量删除消息

您可以删除单条消息，也可以批量删除多条消息。

> **说明**
>
> 删除是高风险操作，删除后数据无法恢复，请谨慎操作。

1. 在消息订阅列表页，选择想要删除的消息后，单击 **删除选中的消息**。
2. 在弹出的窗口中，确认待删除的消息是否正确后，输入待删除的消息条数。
3. 单击 **确认删除**。

3.4. 管理压测流量

消息队列提供压测流量管理功能，是一种消息对象投递策略。消息发布者可以在发送消息时按规则对压测消息进行标记，系统即可自动识别压测消息。压测消息默认不投递到消息订阅方，通过在控制台配置压测流量白名单，压测消息即可被投递至白名单中的订阅者。

压测流量管理的操作步骤如下：

1. 标记压测消息
2. 添加压测流量白名单

标记压测消息

消息发布者需要在本地应用代码中通过配置 `zoneUid` 对压测消息进行标记。`zoneUid` 为两位，如最后一位是英文字母 A ~ J，表示该消息是压测消息。

配置方法示例如下：

```
UniformEvent uniformEvent.setZoneUid("0A") //zoneUid 最后一位是字母 A～J，则表示是压测消息
```

添加压测流量白名单

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **压测流量管理**。
3. 点击 **添加压测流量白名单**，在 添加压测流量白名单 面板，配置以下信息：

   - **消息订阅组**：输入想要加入白名单的消息订阅组。
   - **消息主题**：选择 **全部** 或指定消息主题。
4. 单击 **提交**。

配置完成后，该消息主题下的压测消息将会投递到所配置的消息订阅组。未在指定消息订阅组内的订阅者，则无法接收该消息主题下的压测消息。

4.进阶指南

4.1. 事务型异步消息说明

发送事务型异步消息

事务型消息指发布消息的应用系统在本地数据库事务操作序列中发送的消息。此类消息的投递与数据库事务状态保持一致，当事务状态为“提交”时，消息会被投递到订阅者；当事务状态为“回滚”时，消息不会被投递到订阅者。

事务型消息发布者（Publisher）配置实例如下所示：

```
<sofa:publisher id="mqPublisher" group="P_appname_service">
  <sofa:channels tx-callback="txCallbackListener">
    <sofa:channel value="TP_DEFAULT"/>
  </sofa:channels>
  <sofa:binding.msg_broker/>
</sofa:publisher>

<bean id="txCallbackListener" class="com.antcloud.tutorial.mq.endpoint.service.TxCallbackLi
stener" />
```

事务型消息配置必须设置 `tx-callback` 属性值 `txCallbackListener`，此实现类必须实现 `com.alipay.common.event.UniformEventTxSynMessageListener` 接口，以作为事务型消息回查接口。

当消息代理组件无法确定事务型消息是处于“提交”状态还是“回滚”状态时，会主动调用消息发布者实现的事务型消息回查接口。这种回查属于异常场景，不是每次发送事务型消息都会触发。事务回查接口实现实例如下所示：

```
public class TxCallbackListener implements UniformEventTxSynMessageListener {

    @Override
    public void onUniformEventTxSynchronized(UniformEvent message, UniformEventContext uCon
text) {
        try {
            if (!txMessageCheck(message)) {
                uContext.setRollbackOnly(); // 设置状态为回滚
            }
        } catch (Exception e) {
            throw e; // 抛出异常
        }
    }

    private boolean txMessageCheck(UniformEvent message) {
        return false;
    }
}
```

发布端应用系统接收到事务型消息回查后可能存在以下三种处理策略：

- 设置状态为“回滚”，对应的消息不会投递到订阅者。

- 抛出异常，事务型消息回查会后续重新执行。
- `onUniformEventTxSynchronized` 正常执行完毕，状态为“提交”，对应的消息投递到订阅者。

事务型消息发布者服务实现与 快速开始 文档中描述的普通消息发布者服务实现有以下两点不同：

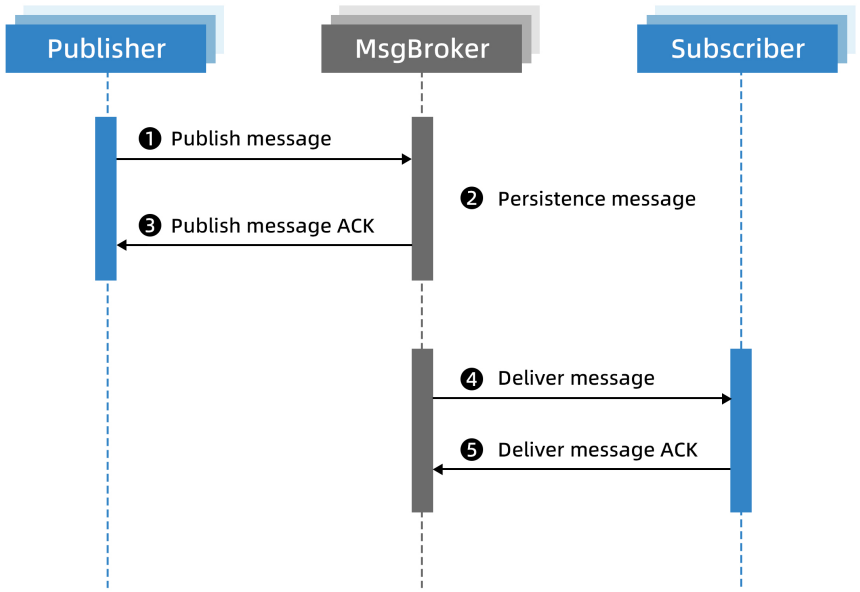
- 必须设置 `uniformEvent.setTransactional(true)` 。
- 必须在 Spring 事务模板中发送消息。具体实现代码如下所示：

```
  public class UniformEventPublisherService {

      private static final Logger logger = LoggerFactory.getLogger(UniformEventPublisherS
ervice.class);

      private UniformEventPublisher uniformEventPublisher;

      private UniformEventBuilder   uniformEventBuilder = new DefaultUniformEventBuilder(
);

      private TransactionTemplate transactionTemplate;

      // 发布事务型消息，消息类型由 topic/eventcode 指定，业务对象作为消息 payload。
      public boolean publicTransactionUniformEvent(String topic, String eventcode, Object
payload) {
          // 创建消息，第一个参数是 topic，第二个参数是 eventcode
          final UniformEvent uniformEvent = uniformEventBuilder.buildUniformEvent(topic,
eventcode);
          // 设置消息负载,一般为业务对象
          uniformEvent.setEventPayload(payload);
          // transactional 为 true 代表事务型消息
          uniformEvent.setTransactional(true);

          transactionTemplate.execute(new TransactionCallback() {
              @Override
              public Object doInTransaction(TransactionStatus status) {
                  try {
                  uniformEventPublisher.publishUniformEvent(uniformEvent);
                  } catch (Exception e) {
                      // 事务型消息状态与本地事务一同回滚
                      status.setRollbackOnly();
                  }
                  return null;
              }
          });
          return true;
      }

      public void setTransactionTemplate(TransactionTemplate transactionTemplate) {
          this.transactionTemplate = transactionTemplate;
      }
  }
```

4.2. 异步消息处理流程

普通消息接收与投递流程

普通消息是相对于事务型消息而言的，普通消息的接收与投递流程如下图所示：



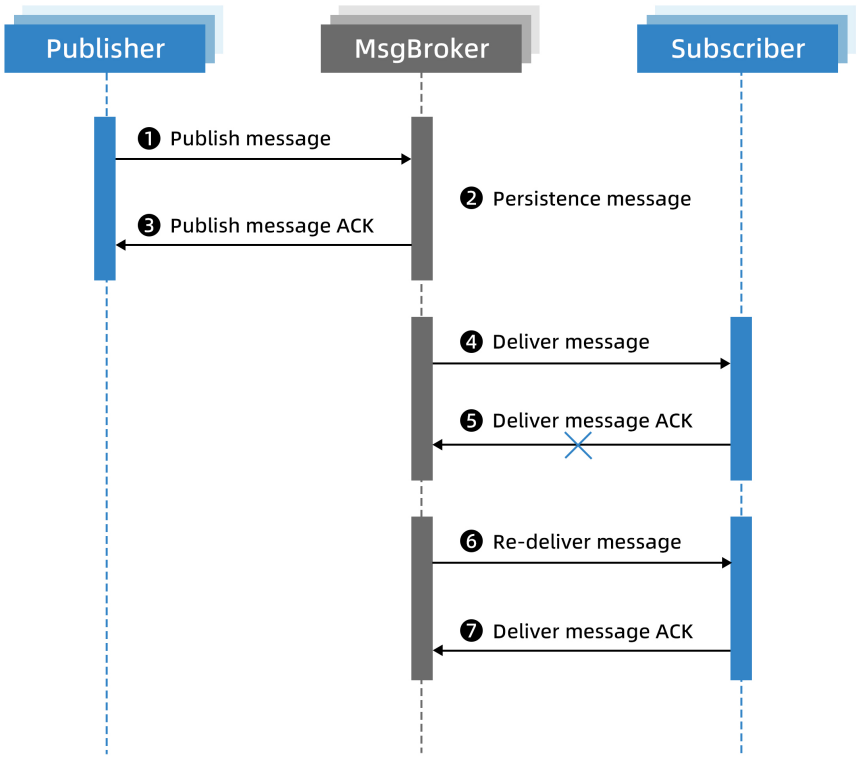
消息重试投递流程

当消息投递异常时，消息代理组件会按照预定时间间隔策略重新投递消息，直至订阅端系统消费成功。

消息投递异常的常见原因如下：

- 订阅端系统未连接到消息代理组件（Message Broker）。
- 订阅端系统收到消息后处理超时，默认消息投递超时时间是10 秒。
- 订阅端系统收到消息后处理异常。
- 订阅端系统收到消息后主动回滚。

消息重试投递流程如下图所示：

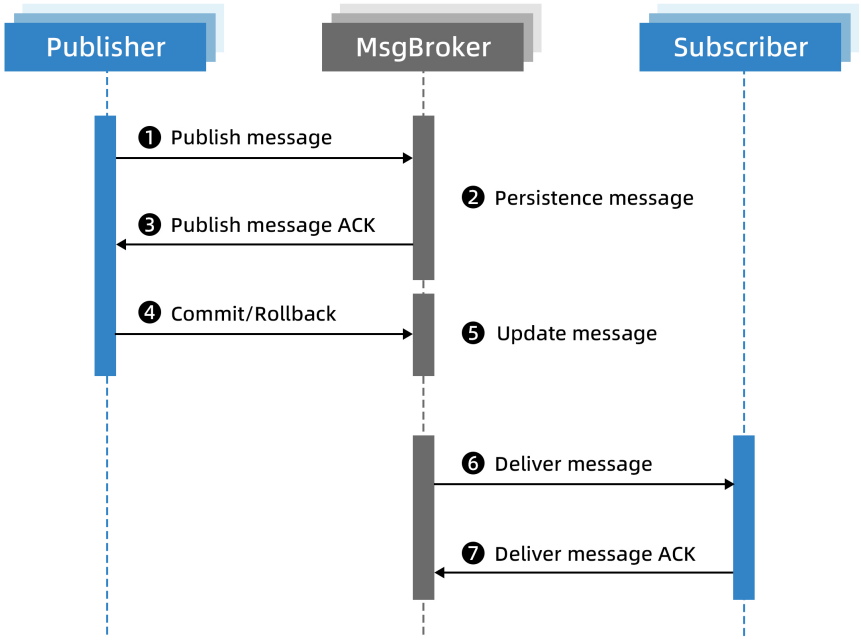


事务型消息接收与投递流程

事务型消息涉及的角色是消息发送端系统和消息代理组件，与消息订阅端系统无关。

事务型消息是否被投递与发送端系统本地数据库事务保持一致，如果本地数据库事务提交则消息会被投递给订阅端；如果本地数据库事务回滚，则直接丢弃消息不投递给订阅端系统。

事务型消息接收与投递流程如下图所示：



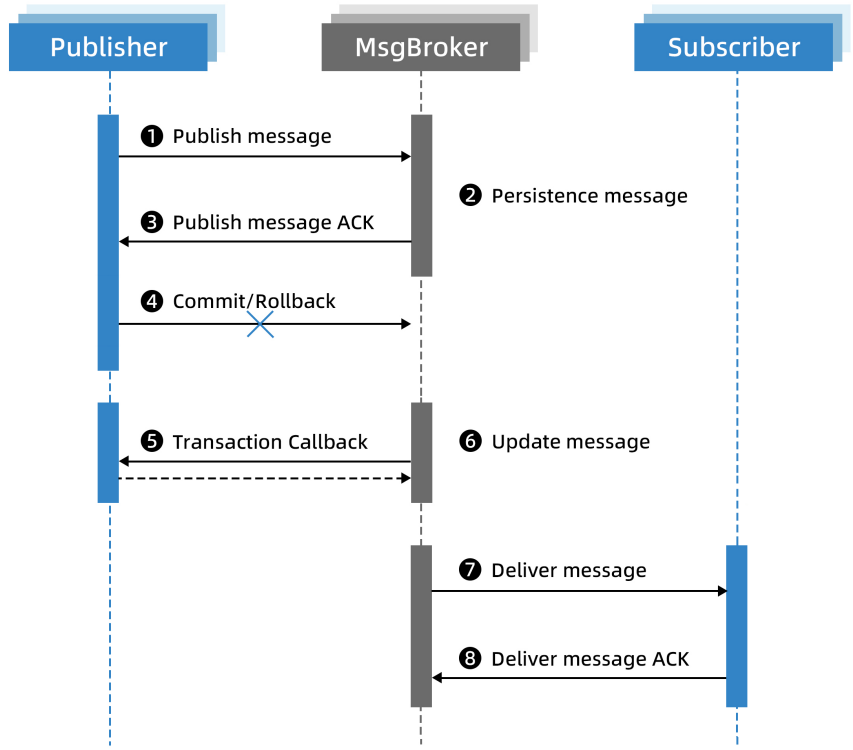
事务型消息回查流程

事务型消息涉及典型的两阶段消息流程：

- **第一阶段**：消息发布端发送消息到消息代理组件。
- **第二阶段**：消息发布端发送提交或者回滚指令到消息代理组件，消息代理组件根据此指令决定是否投递消息到订阅端系统。

当第二阶段指令出现异常时，消息代理组件在一定时间后主动回查消息发送端系统，确认对应的事务型消息是否投递。

事务型消息回查流程如下图所示：



4.3. HEADER 消息订阅类型

消息队列服务支持 DIRECT 和 HEADER 消息订阅类型：

- 当消息订阅类型配置为 DIRECT 时，消息订阅者会收到这一消息类型的全部消息。
- 当消息订阅类型配置为 HEADER 时，您可通过设置自定义过滤表达式对消息进行过滤，以实现只接收符合业务数据条件的部分消息。

添加 HEADER 消息订阅

1. 登录消息队列控制台。
2. 在左侧导航栏上，单击 **消息订阅管理**。
3. 单击 **添加消息订阅**，在 **添加消息订阅** 面板，配置以下信息：

- **应用名称**：本例中输入 `appname` 。
- **消息订阅组**：本例中输入 `S_appname_service` 。
- **消息主题**：本例中输入 `TP_DEFAULT` 。
- **消息事件码**：本例中输入 `EC_DEFAULT` 。
- **持久类型**：本例中选择 **持久型**。
- **消息订阅类型**：选择 **HEADER**。
- **过滤表达式**：过滤表达式支持 `&&` 和 `||` 符号，建议使用英文括号 `( )` 组合表达式。消息队列服务端会根据设置的属性值及过滤表达式的计算结果，决定是否投递消息：
  - 当过滤表达式计算结果为 True 时，消息被投递。
  - 当过滤表达式计算结果为 False 或消息发布者代码中未设置属性值时，消息则不被投递。

  本例中输入 `(property.accountId == '1' || property.accountId == '2')` 。该过滤表达式针对 `accountId` 属性值进行过滤。当 `accountId` 属性值为 1 或 2 时，表达式结果为 True，否则为 False。

4. 单击 **提交**。

消息发布者设置消息属性

在消息发布者代码中使用 `message.setProperty()` 方法设置消息属性值，示例如下：

对于以上过滤表达式示例，代码中可设置 accountId 的属性值，例如：

```
message.setProperty("accountId", "1");
```

此时 `accountId` 属性值为 1，过滤表达式计算结果为 True，消息会被投递。

```
message.setProperty("accountId", "3");
```

此时 `accountId` 属性值为 3，过滤表达式计算结果为 False，消息不被接收。

5.教程

5.1. 发送一条普通消息

本教程的学习目标如下：

- 定义并注册一个合法的消息类型。
- 定义发布者唯一标识 `groupId` 。
- 掌握消息发布者的配置方式。
- 掌握查询消息发送日志的方式。

配置消息类型元数据

消息类型为消息分类的维度，是消息队列产品的一个关键概念。由 TOPIC 和 EVENT CODE 共同唯一标识。通常描述如下：

> **说明**
>
> 发布者 A 发送消息类型 TP_DEFAULT，EC_DEFAULT。

消息类型必须经过注册以后才会生效，否则发送者无法成功发送消息。本教程中使用的 TOPIC 和 EVENT CODE 分别定义为 `TP_DEFAULT` 和 `EC_DEFAULT` 。关于消息类型元数据添加步骤，参见 新增消息类型。

命名发布者唯一标识

发布者为发送消息的应用，是消息中间件的一个关键角色。

发布者的唯一标识为 `groupId` ，命名约定形式为 `P\_appname\_service` 。其中，前缀“P_”代表 Publisher。在本教程中，我们将其命名为 `P_mq_tutorial` 。

发布者 `groupId` 不需要申请，只需要在应用中配置，主要通过约定形式保证唯一性。

添加消息队列依赖

打开 `mqtutorial-publish-message` 根目录下的配置文件 `pom.xml` ，可以看到已经添加好的消息队列依赖：

```
<!-- mq dependency -->
<dependency>
    <groupId>com.alipay.boot</groupId>
    <artifactId>mq-spring-boot-starter</artifactId>
    <version>${mq.version}</version>
</dependency>
```

配置消息发布者

打开配置文件 `mqtutorial-publish-message-endpoint` 查看发布者配置，具体路径如下：

```
mqtutorial-publish-message/app/endpoint/src/main/resources/META-INF/mqtutorial-publish-mess
age/mqtutorial-publish-message-endpoint.xml
```

发布者配置的核心元素是 `sofa:publisher` ，具体内容如下：

```
<!-- Declare a publisher bean with id "mqPublisher" -->
<sofa:publisher id="mqPublisher" group="P_mq_tutorial">
    <sofa:channels>
        <sofa:channel value="TP_DEFAULT"/>
    </sofa:channels>
    <sofa:binding.msg_broker/>
</sofa:publisher>
```

以上配置中有三个关键元素，具体说明如下：

- `id` 属性值是 Spring Bean 唯一标识，用于服务依赖注入。
- `group` 属性值是发布者 group，请修改为 自定义的 groupId。
- `sofa:channel` 元素的 `value` 属性值是需要发送的消息类型 TOPIC 值。

打开源代码文件 `com.antcloud.tutorial.mq.endpoint.service.MqService` ，查看具体发送消息示例。

将 `mqtutorial-publish-message` 项目发布部署到 SOFAStack 平台上。具体发布部署方式，参见 云端发布。

5.2. 订阅一条普通消息

本教程的学习目标如下：

- 定义订阅者唯一标识 `group` 。
- 掌握注册一个有效的消息订阅的方式。
- 掌握配置消息订阅者的的方式。
- 掌握查询消息接收日志的方式。

命名订阅者唯一标识

订阅者（也叫消费者）即订阅消息的应用系统，是消息中间件的一个关键角色。

与发布者相对应，订阅者也需要命名唯一标识称为“订阅者 group”，命名约定格式是 `S\_appname\_service` ，其中“S_”前缀代表 Subscriber。在本教程中，我们将其命名为 `S_appname_service` 。

申请订阅关系

消息订阅是消息队列产品的一个关键概念，包括订阅者唯一标识、消息类型、订阅关系类型和持久化类型四个元素。通常描述如下：

> **说明**
>
> 订阅者 C 订阅了消息类型 TOPIC_DEFAULT，EVENTCODE_DEFAULT；订阅类型是 DIRECT；持久化类型为非持久型。

要订阅到预期的消息类型，订阅者必须申请有效的订阅关系，申请之前需要明确订阅关系的四个必要元素：

- 订阅者唯一标识 `group` ： `S_appname_service` 。
- 需要订阅的目标消息类型 (TOPIC+EVENTCODE)：TP_DEFAULT，EC_DEFAULT。
- 消息订阅类型，默认为 DIRECT。
- 持久化类型，本教程使用“非持久型”。

查询已存在的消息订阅和添加新的消息订阅，参见 查看消息订阅列表 及 添加消息订阅。

配置消息订阅者

目录切换进入 `mqtutorial-subscribe-message` 目录，打开配置文件 `mqtutorial-subscribe-message-endpoint.xml` 查看订阅者配置，具体路径如下：

```
mqtutorial-subscribe-message/app/src/main/resources/META-INF/mqtutorial-subscribe-message/mqtutorial-subscribe-message-endpoint.xml
```

订阅者配置的核心元素是 `sofa:consumer` ，具体内容如下：

```
<!-- Declare a consumer bean with id "mqConsumer" -->
<sofa:consumer id="mqConsumer" group="S_appname_service">
    <sofa:listener ref="mqMessageListener"/>
    <sofa:channels>
        <sofa:channel value="TP_DEFAULT">
            <sofa:event eventType="direct" eventCode="EC_DEFAULT" persistence="true"/>
        </sofa:channel>
    </sofa:channels>
    <sofa:binding.msg_broker/>
</sofa:consumer>

<bean id="uniformEventMessageListener" class="com.alipay.cloud.UniformEventMessageListenerImpl"/>
```

以上配置中有四个关键元素，具体说明如下：

- `sofa:consumer` 元素 `group` 属性值是订阅者 `group` ，请修改为自定义的 `group` 值，即 `S_appname_service` 。
- `sofa:listener` 元素配置了消息接收 Handler。

- `sofa:channel` 元素的 `value` 属性值是需要订阅的消息类型 TOPIC 值。
- `sofa:event` 元素具体声明了订阅关系的其它要素。

首先，打开源代码文件 `com.antcloud.tutorial.mq.endpoint.service.MqMessageListener` 查看具体的消息处理逻辑。示例代码中的消息处理逻辑很简单，只是在接收到消息后输出消息内容日志。需要注意的是，消息处理器必须实现 `com.alipay.common.event.UniformEventMessageListener` 接口。

然后，将 `mqtutorial-subscribe-message` 项目发布部署到 SOFAStack 平台上，具体发布部署方式，参见 云端发布。

6.最佳实践

命名规则

快速开始 中涉及了四个核心概念，包括：TOPIC、EVENTCODE、发布者 GROUP 和消费者 GROUP。其中 TOPIC 和 EVENTCODE 构成消息类型。这四个核心概念的命名规则约束如下所示：

- **TOPIC 命名规则**：以“TP_”为前缀，大写字母组成，下划线分割，长度少于 32 个字符。
- **EVENTCODE 命名规则**：以“EC_”为前缀，大写字母组成，下划线分割，长度少于 64 个字符。
- **发布者 GROUP 命名规则**：以“P_”为前缀，大小写字母组成，长度少于 64 个字符，建议格式为 `P\_appname\_service`，其中 `appname` 是应用系统名称，`service` 是相关服务名称。
- **消费者 GROUP 命名规则**：以“S_”为前缀，大小写字母组成，长度少于 64 个字符，建议格式为 `S\_appname\_service`，其中 `appname` 是应用系统名称，`service` 是相关服务名称。

发送多个消息类型的发布者（Publisher）配置

一个应用系统可能发送多种消息类型，如果涉及多个 TOPIC，允许在一个 `sofa:publisher` 元素中配置多个 `sofa:channel` 元素，每个元素设置一个 TOPIC，而不是配置多个 `sofa:publisher` 实例。配置实例如下所示：

```
<sofa:publisher id="mqPublisher" group="P_appname_service">
  <sofa:channels >
    <sofa:channel value="TP_DEFAULT_A"/>
    <sofa:channel value="TP_DEFAULT_B"/>
  </sofa:channels>
  <sofa:binding.msg_broker/>
</sofa:publisher>
```

> **说明**
>
> 一个应用系统推荐只配置一个 `sofa:publisher` 元素负责消息发布。

订阅多个消息类型的消费者（Consumer）配置

应用系统可能订阅多种消息类型，允许在一个 `sofa:consumer` 元素中配置多个 `sofa:channel` 元素，每个 `sofa:channel` 元素中配置多个 `sofa:event` 元素，而不是配置多个 `sofa:consumer` 实例。在这种场景下，订阅的全部消息由同一个 `uniformEventMessageListener` 实例负责消费，建议按照消息类型维度选择对应的消息处理逻辑。配置实例如下所示：

```
<sofa:consumer id="mqConsumer" group="S_appname_service">
  <sofa:listener ref="mqMessageListener"/>
  <sofa:channels>
    <sofa:channel value="TP_DEFAULT_A">
      <sofa:event eventType="direct" eventCode="EC_DEFAULT_A" persistence="true"/>
      <sofa:event eventType="direct" eventCode="EC_DEFAULT_B" persistence="true"/>
    </sofa:channel>

    <sofa:channel value="TP_DEFAULT_B">
      <sofa:event eventType="direct" eventCode="EC_DEFAULT_C" persistence="false"/>
    </sofa:channel>
  </sofa:channels>
  <sofa:binding.msg_broker/>
</sofa:consumer>

<bean id="mqMessageListener" class="com.antcloud.tutorial.mq.endpoint.service.MqMessageList
ener"/>
```

> **说明**
>
> 一个应用系统推荐只配置一个 `sofa:consumer` 元素负责订阅全部消息类型。

修改默认线程池配置

当消息中心积压超时且应用处理消息能力满足时，可通过修改默认线程池的配置，加大订阅端的吞吐量。配置方法如下：

1. 计算当前订阅端线程池配置。

   每秒处理消息的笔数= 1000 ms / 处理一笔消息的平均耗时 ms

   线程数 = 单机目标 tps / 每秒处理消息的笔数。

   > **说明**
   >
   > 订阅端默认线程池配置是 `core 10 max 20 queueSize 10000` 。

2. 修改当前订阅端线程池配置。

   - 使用 spring 配置

     ```
     UniformEventSubscriberAdapter
     <property name="messageTPCorePoolSize" value="10" />
     <property name="messageTPMaxPoolSize"  value="20" />
     ```

   - 使用 sofa 标签配置

     ```
     <sofa:binding.msg_broker>
           <attributes    messageTPCorePoolSize="100" messageTPMaxPoolSize="120" messageTP
     MaxQueueSize="2000" persistent="true" />
     </sofa:binding.msg_broker>
     ```

3. 完成当前订阅端线程池配置的修改后，重启机器。
4. 检查 `logs/cloudengine/common-default.log` 或者 `sofa-default.log` 日志里的 `CorePoolSize` 和 `MaxPoolSize` 是否发生改变。

```
2016-10-18 10:40:24,138 INFO  msgWorkTP-901400531-1-thread-3   - msgWorkTP-901400531:
CorePoolSize=10
MaxPoolSize=20
PoolSize=10
LargestPoolSize=10
QueueSize=0
QueueRemainingCapacity=10000
TaskCount=363738
ActiveCount=1
CompletedTaskCount=363737
KeepAliveTimeInSeconds=60
avgQSize=721
avgRunTm=118
avgStayInQTm=4601
```

消息订阅持久型类型

消息订阅持久类型包括 **持久型订阅** 和 **非持久性订阅**，具体区别如下：

- **持久型订阅**：当订阅端系统未连接到消息代理组件（Message Broker）时，消息代理组件负责保存消息，待订阅端系统连接后，完成投递。
- **非持久型订阅**：当订阅端系统未连接到消息代理组件（Message Broker）时，消息代理组件直接丢弃需要被投递的消息。此订阅类型一般应用于时效性较强的消息。

7.权限说明

消息队列控制台使用中间件产品的用户角色定义，包括以下三种角色：

- 共享中间件-管理员
- 共享中间件-开发者
- 共享中间件-观察者

具体权限设置请参见下表。若要修改您的用户角色，请联系空间管理员处理。

模块	操作	共享中间件-管理员	共享中间件-开发者	共享中间件-观察者
消息类型管理	查看	√	√	√
	创建	√	√	×
	修改	√	√	×
	删除	√	√	×
	导入导出	√	√	×
消息订阅管理	查看	√	√	√
	创建	√	√	×
	修改	√	√	×
	删除	√	√	×
	导入导出	√	√	×
消息状态查询	按消息 ID 查询	√	√	√
	按消息主题查询	√	√	√
	删除	√	√	×

模块	操作	共享中间件-管理员	共享中间件-开发者	共享中间件-观察者
压测流量管理	查看	√	√	√
	创建	√	√	×
	删除	√	√	×

8.常见问题

常见问题列表：

应用系统发布和订阅消息是否一定需要在云端控制台配置元数据？

是的，作为消息发布者角色的应用系统必须配置“消息类型元数据”并触发生效；作为消息订阅者角色的应用系统必须配置“消息订阅元数据”并触发生效，否则应用系统无法按照预期正常发布和订阅消息。

消息队列 Broker 拒绝接收消息

发送者发送某一消息类型之前必须在消息队列控制台新增消息类型，即 TOPIC 和 EVENTCODE，如果不是已经配置的合法消息类型，消息队列 Broker 会拒绝接收此类型的消息。

发布者发送消息无对应连接

确认是否在消息队列控制台配置了需要发送的消息类型。如果已配置，则再检查 Spring 应用上下文配置的 `sofa:publisher` 元素是否声明了需要发送的消息类型 TOPIC 值；如果有遗漏，就会导致发布者发送消息无对应连接。

如何确认发布者或者订阅者已经连接到消息队列 Broker

消息队列 Broker 服务的网络端口是 **9529**，请查看发布者或者订阅者的部署服务器是否存在到此端口的网络连接。如果没有，则一定没有成功连接到消息队列 Broker，请参见 快速开始 检查发布者和订阅者的 Spring 上下文配置是否正确。

9.基础术语

术语	说明
消息中间件	在分布式系统环境中，支持从发布者系统接收消息并投递到订阅者系统的软件组件。
消息代理组件（Message Broker）	支持消息高可靠特性的消息中间件。
消息发布者（Publisher）	发送消息的应用系统。
消息订阅者（Subscriber）	接收一种或者多种消息的应用系统，也可叫做消息消费者（Consumer）。
Topic	代表一个消息大类，表示一类具体的业务（并不是表示一个系统）。
Eventcode	代表一个消息大类（Topic）下的一个消息子类，用来表述一个大的业务下具体的功能。
pub groupId	一类 publisher 的标识，这类 publisher 通常生产并发送一类消息，且发送逻辑一致。
sub groupId	一类 subscriber 的标识，这类 subscriber 通常接收并消费一类消息，且消费逻辑一致。
事务消息	消息中心提供类似 X/Open XA 的分布事务功能，并保证消息与本地数据库事务一致性。
订阅关系	订阅关系表示订阅方的 GroupId 和消息类型的映射关系，是消息中心投递消息的唯一依据。