Message Queue for Apache RocketMQ

SDK Reference

MORE THAN JUST CLOUD | Alibaba Cloud

SDK Reference

SDK guide (TCP)

Java SDK

Release Notes

This topic provides the download links, versions, and updates of all Java SDKs so that you can choose a suitable one for use.

ons-client v1.8.4.Final

Release date	Version	Download	Download (including the Exactly-Once delivery semantics)	Environment preparation guide
2019-09-27	1.8.4.Final	ons-client-1.8.4.Final	ons-client-ext-1.8.4.Final	Prepare the Java SDK environment

New features

- Supported 1.6 JDK ;
- Supported retries of asynchronous message sending failures;
- Supported retries of synchronous message sending failures due to brokerbusy.

ons-client v1.8.0. Final

Release date	Version	Download	Download	Environment

			(including the Exactly-Once delivery semantics)	preparation guide
2019-02-21	1.8.0.Final	ons-client-1.8.0.Final	ons-client-ext-1.8.0.Final	Prepare the Java SDK environment

Bugs fixed

- Fixed the automatic retry logic when a message failed to be synchronized from the producer to a new topic on an instance after MQ instantiation is implemented. A maximum of three retries are supported by default.

ons-client v1.7.8.Final

Release date	Version	Download	Environment preparation guide
2018-07-06	1.7.8.Final	ons-client-1.7.8.Final	Prepare the Java SDK environment

New features

- Supported dynamic STS token update.

Bugs fixed

- Changed the default size of logs from 1 GB to 64 MB.
- Fixed the problem of double printing logs.

More historical versions

ons-client v1.7.7.Final

Release date	Version	Download	Environment preparation guide
2018-04-25	1.7.7.Final	ons-client-1.7.7.Final	Prepare the Java SDK environment

Bugs fixed

- Fixed the problem that trajectorial messages cannot be sent when multiple consumer/producer instances are initialized in one process (in version 1.7.5 or 1.7.6).

ons-client v1.7.6.Final

Release date	Version	Download	Environment preparation guide
2018-04-04	1.7.6.Final	ons-client-1.7.6.Final	Prepare the Java SDK environment

New features

- Supported the compatibility with any log framework.

Bugs fixed

- Provided support for Log4j2.
- Fixed the problem of client fetchNameserver shutdown.
- Upgraded the Fastjson version to 1.2.48.

ons-client v1.7.5.Final

Release date	Version	Download	Environment preparation guide
2018-03-23	1.7.5.Final	ons-client-1.7.5.Final	Prepare the Java SDK environment

Bugs fixed

- Fixed the problem of introducing internal dependencies of Alibaba.

ons-client v1.7.4.Final

Release date	Version	Download	Environment preparation guide
2018-03-02	1.7.4.Final	ons-client-1.7.4.Final	Prepare the Java SDK environment

New features

- Supported STS token access.
- Prioritized trajectorial message transmission. Such messages are sent to the MQ broker of the current cluster in prior by default.

Bugs fixed

- Fixed the JDK 1.6 incompatibility problem.

ons-client v1.7.2.Final

Release date	Version	Download	Environment preparation guide
2018-01-25	1.7.2.Final	ons-client-1.7.2.Final	Prepare the Java SDK environment

New features

- Encrypted transmission of the AcessKeyId/AccessKeySecret signature chain is supported in the Enterprise Platinum Edition, improving data security.
- Supported SQL attribute filtering for consumers in the Enterprise Platinum Edition, greatly improving the efficiency of message subscription.
- Added the feature that the client automatically senses the changes of NameServer, which facilitates O&M switching and ensures high availability.
- Added the function of reporting the exact version of a newly connected client.

ons-client v1.7.1.Final

Release date	Version	Download	Environment preparation guide
2017-12-19	1.7.1.Final	ons-client-1.7.1.Final	Prepare the Java SDK environment

New features

- Added an asynchronous sending operation for customizing callback thread pool.
- Added a JVM-D parameter to the asynchronous sending operation, which is used to control the number of threads in the public thread pool: Dclient.callback.executor.thread.nums=10.

Bugs fixed

- Fixed the problem that the cache count in SendBack is not subtracted when consumer consumption times out.
- Fixed the problem of the premature release of the consumer asynchronous signals.

ons-client v1.7.0.Final

Release date	Version	Download	Environment preparation guide
2017-10-23	1.7.0.Final	ons-client-1.7.0.Final	Prepare the Java SDK environment

New features

- Adjusted the client message cache policies in two dimensions: number of messages and cache size.

Function optimization

- Optimized ProducerName of the built-in trace module of the client to apply different values to different users.

Bugs fixed

- Fixed the problem that a client trace thread blocks the client from exiting normally.
- Fixed the problem that the message trace ShutDownHook is created repetitively.

ons-client v1.6.1.Final

Release date	Version	Download	Environment preparation guide
2017-08-31	1.6.1.Final	ons-client-1.6.1.Final	Prepare the Java SDK environment

Function optimization

- Added details Java documentation for all client APIs.
- Optimized the mode for obtaining client addresses, which is not dependent on the hostname configuration in /etc/hosts.

ons-client v1.6.0.Final

Release date	Version	Download	Environment preparation guide
2017-07-31	1.6.0.Final	ons-client-1.6.0.Final	Prepare the Java SDK environment

New features

- Shaded the client at the source code level to ensure correct debugging.
- Exposed message attributes BornHost and BornTimestamp.
- Added the BatchConsumer operation to allow users to consume messages in batches.
- Added the demo for integrating ordered messages, BatchConsumer, and Spring.

Function optimization

- Placed Sharding Key in the message structure for partitionally ordered messages.
- Supported integer values for message attribute settings.

Demo project (TCP)

This demo helps engineers who are new to MQ to build a MQ test project step by step. The demo program provides test codes in Java mode for normal messages, transactional messages, and scheduled messages, as well as Spring configuration.

Prepare the environment

Step 1: Install IDE

You can use IntelliJ IDEA or Eclipse. IntelliJ IDEA is used in this example.

Download IntelliJ IDEA Ultimate from https://www.jetbrains.com/idea/ and install it according to the instructions.

Step 2: Download the demo project

Download the demo project from https://github.com/AliwareMQ/mq-demo to a local machine and decompress the .zip file. Then, a folder named Aliware-MQ-demo-master is created on the local machine.

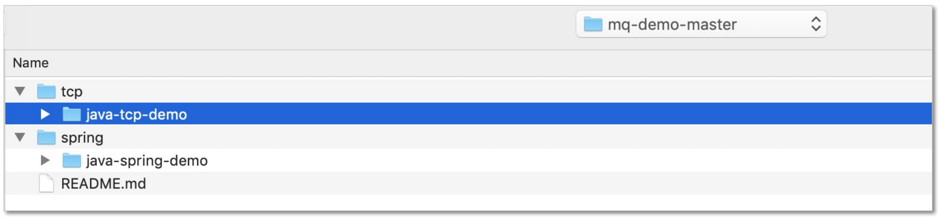
Set the demo project as follows:

Configure the demo project

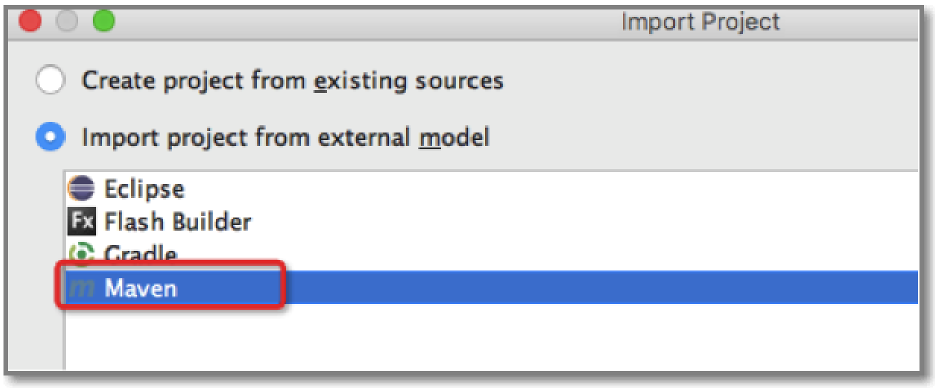
Step 1: Import the demo project to IntelliJ IDEA

Prerequisites: You have installed the JDK on your local machine. If not, download and install it first.

In IntelliJ IDEA, select **Import Project**, and then select the folder **mq-demo-master**.



Select Import project from external model.


Import Project
Create project from existing sources
Import project from external model
Eclipse
Flash Builder
Gradle
Maven

Click Next until the project is imported. The dependent JAR package needs to be loaded to the demo project. Therefore, it takes two to three minutes to import the project.

Step 2: Create resources

Create the required resources in the MQ console, including a MQ instance, a topic, a group ID, and an AccessKey for authentication.

For more information and operation instructions, see **Step 2: Create resources** in Quick start for primary accounts.

Step 3: Configure the demo

Configure two files: MqConfig class and common.xml.

1. Configure the MqConfig class as follows:

- public static final String TOPIC = "the topic you created" ;
- public static final String GROUP_ID = "the group ID you created" ;
- public static final String ACCESS_KEY = "the AccessKeyId of your Alibaba Cloud account" ;
- public static final String SECRET_KEY = "the AccessKeySecret of your Alibaba Cloud account" ;
- public static final String NAMESRV_ADDR = "the TCP endpoint of your MQ instance that is displayed as **TCP Endpoint** in the **Endpoint Information** area on the **Instances** page" ;

Note:

- For more information about how to create an AccessKey (including an AccessKeyId and an AccessKeySecret), see Create an AccessKey.
- You can also use the AccessKey of the RAM sub-account if it is granted with the permissions of the topic you created.

1. Configure producer.xml.

```
```

```
  <props>
<prop key="AccessKey">XXX</prop>  <!-- enter values for these parameters -->
<prop key="SecretKey">XXX</prop>
<prop key="GROUP_ID">XXX</prop>
<prop key="Topic">XXX</prop>
<prop key="NAMESRV_ADDR">XXX</prop>
</props>
```

1. Configure consumer.xml.

```


<bean id="consumer" class="com.aliyun.openservices.ons.api.bean.Consumerbean"
init-method="start" destory-method="shutdown">
<property name="properties">
<map>
<entry key="GROUP_ID" value="XXX"/><!——Replace GROUP_ID with the group ID of the consumer.——>
<entry key="AccessKey" value="XXX"/><!——Replace AccessKey with the AccessKeyId of your account.——>
<entry key="SecretKey" value="XXX"/><!——Replace SecretKey with the AccessKeySecret of your account.——>
<entry key="ONSAddr" value="http://onsaddr-internet.aliyun.com/MQ/nsaddr4client-internet"/>
</map>
</property>

<property name="subscriptionTable">
<map>
<entry value-ref="messageListener">
<key>
<bean class="com.aliyun.openservices.ons.api.bean.Subscription">
<property name="topic" value="XXX"/><!——Replace the topic with the one you created.——>
<property name="expression" value="*"/><!——The name of the message type. Separate multiple message types
with "||".——>
</bean>
</key>
</entry>
</map>
</property>
</bean>
```

Run the demo project

Start sending and receiving messages in Main mode

Execute SimpleMQProducer to send messages.

Log on to the MQ console. In the left-side navigation pane, choose **Message Query** > **By Topic**. On the page that is displayed, select the topic you want to query. The query result indicates that the messages have been sent to the topic.

Execute SimpleMQConsumer to receive messages. The log indicating that the messages are received is printed. The class needs to be initialized, which takes several seconds. Initialization does not often occur in the production environment.

In the MQ console, choose **Consumers** > **Consumer Status**. The system shows that the started consumer is online and the subscriptions are consistent.

Start sending and receiving messages in Spring mode

1. Execute ProducerClient to send messages.
2. Execute ConsumerClient to receive messages.

Perform similar operations as above to view the results.

Send transactional messages

Execute SimpleTransactionProducer to send messages.

Note: LocalTransactionCheckerImpl is an API that you can call to check local transactions. It is used for verifying transactions. For more information, see Send and receive transactional messages.

Send and receive ordered messages

Execute SimpleOrderConsumer to receive messages.

Execute SimpleOrderProducer to send messages.

Note: In this mode, messages are delivered and consumed in order. For more information, see Send and receive ordered messages.

Send scheduled/delayed messages

Execute MQTimerProducer to send messages. Messages are delivered in 3s.

Note: You can specify an exact delivery time, which is up to 40 days. For more information, see Send and receive scheduled messages.

How to use the Java SDK

MQ provides the Java SDK for message delivery and subscription. This topic describes the parameters of Java interfaces and shows you how to use these interfaces.

Sample code for the transmission and reception of messages

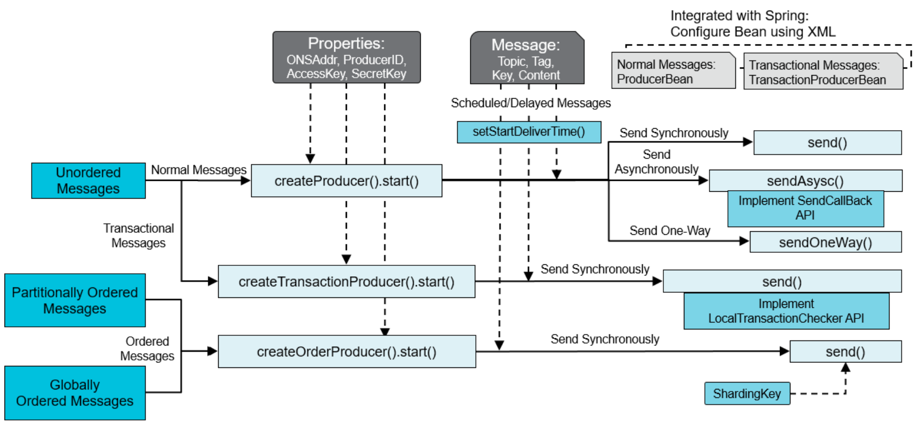
- Send and receive normal messages
- Send and receive ordered messages
- Send and receive scheduled messages
- Send and receive delayed messages
- Send and receive transactional messages

Common parameters

Parameter	Description
NAMESRV_ADDR	The TCP endpoint, which is obtained from the console.
AccessKey	The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
SecretKey	The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
OnsChannel	The user channel, which is ALIYUN by default and CLOUD for Alibaba Retail Cloud users.

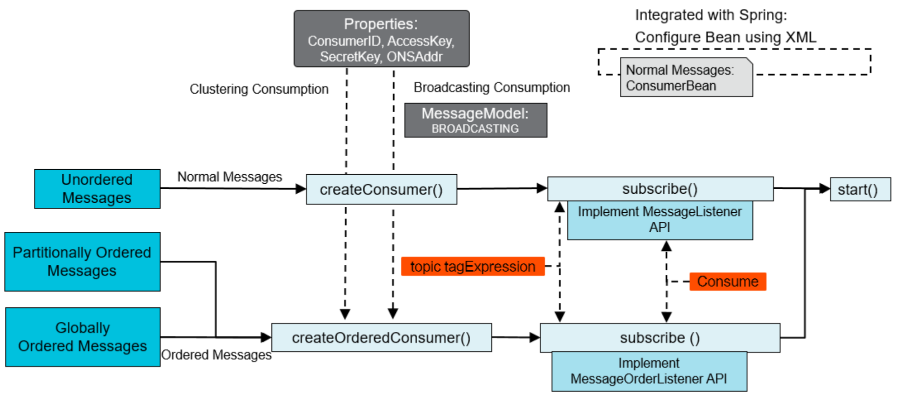
Message transmission parameters

Parameter	Description
SendMsgTimeoutMillis	The message transmission timeout period (in milliseconds). Default value: 3000
CheckImmunityTimeInSeconds (transactional messages)	The shortest time interval (in seconds) before the first recheck of a transactional message.
shardingKey (ordered messages)	The sharding key for ordered messages.



Message subscription parameters

Parameter	Description
GROUP_ID	The group ID you created in the console.
MessageModel	The consumption mode of a consumer instance, which can be CLUSTERING (default) or BROADCASTING.
ConsumeThreadNums	The number of consumption threads for a consumer instance. Default value: 20
MaxReconsumeTimes	The maximum number of retries upon a consumption failure. Default value: 16
ConsumeTimeout	The maximum consumption timeout period for each message. If a message fails to be processed within this period, the consumption fails, and the message needs to be resent for consumption. Set an appropriate value (in minutes) for each business application. Default value: 15.
suspendTimeMillis (ordered message)	The retry interval for messages that fail to be consumed.



Java SDK preparation

Before executing the Java code described in this section, please prepare the environment according to the following instructions.

Dependency can be introduced through either one of the following two methods:

Introduce dependency using Maven:

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>ons-client</artifactId>
<version>"XXX"</version>
//version number of the latest Java SDK
</dependency>
```

About the version number of the latest Java SDK, see Release notes.

Download a dependency JAR package:

About the download link of the latest Java SDK, see Release notes.

The topic and consumer ID involved in the code need to be created first in the MQ console. For details on how to create topics and consumer ID, see **Step 2: Create resources** in Quick start guide.

Applications using MQ services through TCP need to be deployed on ECS instances in the same region.

Log configuration

This topic describes how to print the logs of the MQ client and to customize client log configuration.

Print client logs

Client logs are important for problem location. Logs record exceptions during operation of the MQ client, helping to reproduce the exception that occurs at a specific time. You can locate and fix system bugs by checking logs.

The TCP Java SDK of MQ is programmed by using Simple Logging Facade for Java (SLF4J).

Java SDK 1.7.8. Final or later

Java SDK 1.7.8.Final for MQ has a built-in logging framework. Therefore, you can print logs of the MQ client without adding the dependency of logging framework to the application.

For information about the default log configuration for the MQ client and how to modify the default configuration, see Customize log configuration below.

Java SDK earlier than 1.7.8. Final

Java SDK earlier than 1.7.8.Final for MQ only supports log4j and logback other than log4j2. For these versions, you must add the dependency of logging framework to the pom.xml configuration file or lib file before you can print logs of the MQ client.

The sample code for adding the dependency of log4j or logback is as follows.

Method 1: Depend on the log4j logging framework

```
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>jcl-over-slf4j</artifactId>
<version>1.7.7</version>
</dependency>
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>1.7.7</version>
</dependency>
<dependency>
<groupId>log4j</groupId>
<artifactId>log4j</artifactId>
<version>1.2.17</version>
</dependency>
```

Method 2: Depend on the logback logging framework

```
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-core</artifactId>
<version>1.1.2</version>
</dependency>
```

```
<dependency>
<groupId>ch.qos.logback</groupId>
<artifactId>logback-classic</artifactId>
<version>1.1.2</version>
</dependency>
```

Note: If an application depends on both log4j and logback, the client logs cannot be printed properly due to log conflicts. To print client logs properly, ensure that the application depends only on one logging framework. We recommended that you run the mvn clean dependency: tree | grep log command to check the dependency of logging framework.

Customize log configuration

The MQ client allows you to set the **log storage path**, **log level**, and **maximum number of historical log files retained**. To facilitate log transmission and reading, MQ does not allow you to change the size of each log file and retains the default value of 64 MB.

These parameters are described as follows:

- Log storage path: Ensure that the application has the write permission for this path. Otherwise, logs cannot be printed.
- Maximum number of historical log files retained: You can set this parameter to a value in the range from 1 to 100. If you enter a value beyond this range or in an invalid format, the system retains 10 historical log files by default.
- Log level: The system supports logs of the ERROR, WARN, INFO, and DEBUG levels. If an incorrect value is entered, the system retains the default value INFO.

Default configuration

After you start the client, the client generates log files based on the following default configuration:

- Log storage path: /{user.home}/logs/ons.log, where {user.home} is the root directory of the account that runs the current Java process
- Maximum number of historical log files retained: 10
- Log level: INFO
- Size of a single log file: 64 MB

Custom configuration

Note: To customize the log configuration for the MQ client, upgrade the Java SDK to 1.2.5 or later.

To customize the log configuration in the Java SDK, configure the following system parameters:

- ons.client.logRoot: log storage path
- ons.client.logFileMaxIndex: maximum number of historical log files that are retained
- ons.client.logLevel: log level

Example

Add the following system parameters to the startup script or IDE VM options:

```
-Dons.client.logRoot=/home/admin/logs -Dons.client.logLevel=WARN -Dons.client.logFileMaxIndex=20
```

Spring integration

This topic introduces how to send and receive messages using MQ in Spring framework. This section mainly include 3 parts: integration of normal message producer and Spring, integration of transactional message producer and Spring, and integration of message consumer and Spring.

Make sure the subscriptions of all consumer instances under the same consumer ID are consistent. For detailed information, see Subscription consistency.

Configuration parameters supported in Spring framework are the same with those in TCP Java. For detailed information, see Java SDK introduction.

For more information about the Java SDK versions, see Release Notes.

Integration of Producer and Spring

Define information such as producer Bean in producer.xml.

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="producer" class="com.aliyun.openservices.ons.api.bean.ProducerBean" init-method="start"
destroy-method="shutdown">
<!-- All configuration items supported in Java SDK are also supported in Spring method. -->
<property name="properties" > <!--Producer configuration-->
<props>
<prop key="AccessKey">XXX</prop>
<prop key="SecretKey">XXX</prop>
</props>
</property>
</bean>

</beans>
```

Produce messages through the producer that has been integrated with Spring.

```
 package demo;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.exception.ONSClientException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ProduceWithSpring {
public static void main(String[] args) {
/**
* The producer Bean is configured in producer.xml, which can be obtained through ApplicationContext
or directly injected to other classes (such as specific controller).
*/
ApplicationContext context = new ClassPathXmlApplicationContext("producer.xml");

Producer producer = (Producer) context.getBean("producer");
//Send messages in loop
for (int i = 0; i < 100; i++) {
Message msg = new Message( //
// The topic of the message
"TopicTestMQ",
//Message tag, which is similar to tag in Gmail, and is used to classify messages. Consumers can then set
filtering conditions for messages to be filtered in MQ broker.
"TagA",
// Message body, which can be any data in binary format.
// Serialization and deserialization methods need to be negotiated and remain consistent between the
producer and the consumer.
"Hello MQ".getBytes());
// The setting represents the key business property of the message, so please keep it globally unique.
// You can query a message and resend it through the MQ console when you cannot receive the
message properly.
// Note: Message sending and receiving is not affected if you do not configure this setting.
msg.setKey("ORDERID_100");
// Synchronous message sending will succeed as long as no exception is thrown.
try {
SendResult sendResult = producer.send(msg);
assert sendResult != null;
System.out.println("send success: " + sendResult.getMessageId());
}catch (ONSClientException e) {
System.out.println( “Message sending fails");
}

}
}
}
```

Integration of Transactional Message Producer and Spring

For the concept of transactional messages, see Send and receive transactional messages.

Implement LocalTransactionChecker, as shown below. A message producer can have only one LocalTransactionChecker.

```
 package demo;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionChecker;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;

public class DemoLocalTransactionChecker implements LocalTransactionChecker {
public TransactionStatus check(Message msg) {
System.out.println("Start to check status of the local transaction");
return TransactionStatus.CommitTransaction; //Return different transaction status according to the check
result
}
}
```

Define information such as transactional message producer Bean in transactionProducer.xml.

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="localTransactionChecker" class="demo.DemoLocalTransactionChecker"></bean>

<bean id="transactionProducer" class="com.aliyun.openservices.ons.api.bean.TransactionProducerBean"
init-method="start" destroy-method="shutdown">
<property name="properties" > <!--Transactional message producer configuration-->
<props>
<prop key="AccessKey">AKDEMO</prop>
<prop key="SecretKey">SKDEMO</prop>
</props>
</property>
<property name="localTransactionChecker" ref="localTransactionChecker"></property>
</bean>

</beans>
```

Produce transactional messages through the producer that has been integrated with Spring.

```
 package demo;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ProduceTransMsgWithSpring {

public static void main(String[] args) {
/**
* The transactional message producer Bean is configured in transactionProducer.xml, which can be
obtained through ApplicationContext or directly injected to other classes (such as specific controller).
* Refer to the example "Send Transactional Messages"
*/
ApplicationContext context = new ClassPathXmlApplicationContext("transactionProducer.xml");

TransactionProducer transactionProducer = (TransactionProducer)
context.getBean("transactionProducer");

Message msg = new Message("XXX", "TagA", "Hello MQ transaction===".getBytes());

SendResult sendResult = transactionProducer.send(msg, new LocalTransactionExecuter() {
@Override
public TransactionStatus execute(Message msg, Object arg) {
System.out.println("Execute local transaction");
return TransactionStatus.CommitTransaction; //Return different transaction status according to the
execution result of local transaction
}
}, null);
}
}
```

Integration of Consumer and Spring

Create MessageListener, as shown below.

```
 package demo;

import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;

public class DemoMessageListener implements MessageListener {

public Action consume(Message message, ConsumeContext context) {

System.out.println("Receive: " + message.getMsgID());
try {
//do something..
return Action.CommitMessage;
}catch (Exception e) {
//Consumption fails
return Action.ReconsumeLater;
}
}
}
```

Define information such as consumer Bean in consumer.xml.

```
 <?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="msgListener" class="demo.DemoMessageListener"></bean> <!--Listener configuration-->
<!-- You can create multiple ConsumerBeans for multiple CIDs to subscribe to the same topic-->
<bean id="consumer" class="com.aliyun.openservices.ons.api.bean.ConsumerBean" init-method="start"
destroy-method="shutdown">
<property name="properties" > <!--Consumer configuration-->
<props>
<prop key="ConsumerId">CID_DEMO</prop> <!--Please replace XXX-->
<prop key="AccessKey">AKDEMO</prop>
<prop key="SecretKey">SKDEMO</prop>
<!--Fix the number of consumer threads at 50
<prop key="ConsumeThreadNums">50</prop>
-->
</props>
</property>
<property name="subscriptionTable">
<map>
<entry value-ref="msgListener">
<key>
<bean class="com.aliyun.openservices.ons.api.bean.Subscription">
<property name="Topic" value="TopicTestMQ"/>
<property name="expression" value="*"/><!--” expression” is the tag, which can be set to a specific tag,
such as taga||tagb||tagc, or can be set to *. * means subscribing to all tags, and wildcards are not
supported-->
</bean>
</key>
</entry>
<!--For more subscriptions, you can add entry nodes, as shown below-->
<entry value-ref="msgListener">
<key>
<bean class="com.aliyun.openservices.ons.api.bean.Subscription">
<property name="Topic" value="TopicTestMQ-Other"/> <!--subscribe to another topic -->
<property name="expression" value="Taga||Tagb"/> <!-- subscribe to multiple tags -->
</bean>
</key>
</entry>
</map>
</property>
</bean>
```

```
</beans>
```

Run the consumer that has been integrated with Spring, as shown below.

```
 package demo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class ConsumeWithSpring {
public static void main(String[] args) {

/**
* The consumer Bean is configured in consumer.xml, which can be obtained through ApplicationContext
or directly injected to other classes (such as specific controller).
*/
ApplicationContext context = new ClassPathXmlApplicationContext("consumer.xml");
System.out.println("Consumer Started");
}
}
```

Exactly-Once delivery semantics

This topic descibes how to send and receive messages through Exactly-Once delivery semantics. By doing so, the final processing result of a message is written to the database for once only.

Note: Currently, the Exactly-Once delivery semantics is only supported in Java SDK. For more information about the SDK download, refer to Release Notes.

Background Information

The Exactly-Once delivery semantics of MQ applies to the following process: **receive a message > process the message > persistently store the result in database**. It ensures that the final consumption result of each message is written to your database only once, keeping message consumption idempotent.

Procedure

To use the Exactly-Once delivery semantics, follow these steps:

Add dependencies of the SDK package and Spring 3.0 or later to the application. For more information, see Step 1: Add dependencies.

Create a table named transaction_record in the database that stores message consumption results. For more information, see Step 2: Create a consumption transaction table.

Note: The database system that stores message consumption results must support local transactions.

On the message producer, set the PropertyKeyConst.EXACTLYONCE_DELIVERY attribute to enable the Exactly-Once delivery semantics. For more information, see Step 3: Enable the Exactly-Once delivery semantics on the producer.

Create an Exactly-Once consumer on the consumer client and enable Exactly-Once consumption. For more information, see Step 4: Enable the Exactly-Once delivery semantics on the consumer.

Step 1: Add dependencies

The Exactly-Once consumer feature of MQ is released in SDK ons-client-ext-1.8.4.Final . To use the Exactly-Once delivery semantics, add the dependency of this SDK in the application.

In addition, the Exactly-Once consumer depends on Spring to enable Exactly-Once consumption by using the annotation @MQTransaction. Therefore, you must also add the dependency of Spring 3.0 or later in the application.

Add the dependencies as follows:

```
<dependency>
<groupId>com.aliyun.openservices</groupId>
<artifactId>ons-client-ext</artifactId>
<version>1.8.4.Final</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>${spring-version}</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-jdbc</artifactId>
<version>${spring-version}</version>
</dependency>
```

Step 2: Create a consumption transaction table

To use the Exactly-Once delivery semantics of MQ, create a table named transaction_record in the database that persistently stores business processing results. Ensure that this table is in the same database as the table that stores business processing results, and that the database supports local transactions.

The Exactly-Once delivery semantics of MQ support access to MySQL and SQL Server data sources. The statements for creating the transaction_record table for the two types of data sources are as follows.

- MySQL

```
CREATE TABLE `transaction_record` (
`consumer_group` varchar(128) NOT NULL DEFAULT '',
`message_id` varchar(255) NOT NULL DEFAULT '',
`topic_name` varchar(255) NOT NULL DEFAULT '',
`ctime` bigint(20) NOT NULL,
`queue_id` int(11) NOT NULL,
`offset` bigint(20) NOT NULL,
`broker_name` varchar(255) NOT NULL DEFAULT '',
`id` bigint(20) NOT NULL AUTO_INCREMENT,
PRIMARY KEY (`id`),
UNIQUE KEY `message_id_unique_key` (`message_id`),
KEY `ctime_key` (`ctime`),
KEY `load_key` (`queue_id`,`broker_name`,`topic_name`,`ctime`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- SQL Server

```
CREATE TABLE transaction_record
(
[consumer_group] varchar(128) NOT NULL ,
[message_id] varchar(255) NOT NULL ,
[topic_name] varchar(255) NOT NULL ,
[ctime] bigint NOT NULL ,
[queue_id] int NOT NULL ,
[offset] bigint NOT NULL ,
[broker_name] varchar(50) NOT NULL ,
[id] bigint IDENTITY(1,1) PRIMARY KEY
)
CREATE NONCLUSTERED INDEX load_key ON transaction_record
(queue_id, broker_name, topic_name, ctime);
CREATE UNIQUE NONCLUSTERED INDEX message_id_uniq_key ON transaction_record
(message_id);
CREATE NONCLUSTERED INDEX ctime_key ON transaction_record
(ctime);
```

Note:

If you are using Enterprise SQL Server, we recommend that you run ALTER DATABASE [USERDB] SET PARTNER SAFETY OFF; to enable the asynchronous mode, which improves the database read and write performance.

You can also enable the delayed durability feature for the SQL Server database by running ALTER DATABASE [USERDB] SET DELAYED_DURABILITY=FORCED. This feature reduces the IOPS of the database.

Step 3: Enable the Exactly-Once delivery semantics on the producer

On the producer, set the PropertyKeyConst.EXACTLYONCE_DELIVERY attribute to true to enable the Exactly-Once delivery semantics. The sample code is as follows:

```
/**
* Start TestExactlyOnceProducer.
* Set the PropertyKeyConst.EXACTLYONCE_DELIVERY attribute to enable the Exactly-Once delivery semantics.
*/

public class TestExactlyOnceProducer {
public static void main(String[] args) {
Properties producerProperties = new Properties();
producerProperties.setProperty(PropertyKeyConst.GROUP_ID, "{gid}");
producerProperties.setProperty(PropertyKeyConst.AccessKey, "{accessKey}");
producerProperties.setProperty(PropertyKeyConst.SecretKey, "{secretKey}");
producerProperties.setProperty(PropertyKeyConst.NAMESRV_ADDR, "{NAMESRV_ADDR}");
producerProperties.setProperty(PropertyKeyConst.EXACTLYONCE_DELIVERY, "true");
Producer producer = ExactlyOnceONSFactory.createProducer(producerProperties);
producer.start();
System.out.println("Producer Started");

for (int i = 0; i < 10; i++) {
Message message = new Message("{topic}", "{tag}", "mq send transaction message test".getBytes());
try {
SendResult sendResult = producer.send(message);
assert sendResult ! = null;
System.out.println(new Date() + " Send mq message success! msgId is: " + sendResult.getMessageId());
} catch (ONSClientException e) {
System.out.println("Sending failure");
// This exception indicates a message sending failure. To prevent message loss, we recommend that you cache the
message and try again.
}
}
producer.shutdown();
}
}
```

Step 4: Enable the Exactly-Once delivery semantics on the consumer

When the Exactly-Once delivery semantics of MQ is used for consumption, you need to create an Exactly-Once consumer on the consumer client by using ExactlyOnceONSFactory to call the createExactlyOnceConsumer operation. The Exactly-Once consumer then implements Exactly-Once consumption.

For an Exactly-Once consumer, pay attention to the following:

- When creating an Exactly-Once consumer, set the PropertyKeyConst.EXACTLYONCE_DELIVERY attribute to enable or disable the Exactly-Once delivery semantics. Exactly-Once delivery semantics is enabled for an Exactly-Once consumer by default.

- When an Exactly-Once consumer is used for consumption, your business processing logic needs to use MQDataSource in the consume method of the message listener to read and write data in the database.

Use any of the following methods to enable the Exactly-Once delivery semantics on the consumer:

- Enable Exactly-Once delivery semantics without using Spring

- Create transactions on the message listener for database operations and message consumption

- Use the Spring Boot annotation to enable the Exactly-Once delivery semantics on the message listener

- Use MyBatis to enable the Exactly-Once delivery semantics on the message listener

Enable Exactly-Once delivery semantics without using Spring

Example:

```
/**
* Start the Exactly-Once consumer.
* Set the PropertyKeyConst.EXACTLYONCE_DELIVERY attribute to enable the Exactly-Once delivery semantics.
*/
public class TestExactlyOnceConsumer {
private ExactlyOnceConsumer consumer;
private TxMessageListener listener;

public TestExactlyOnceConsumer() {
Properties consumerProperties = new Properties();
consumerProperties.setProperty(PropertyKeyConst.GROUP_ID, "{gid}");
consumerProperties.setProperty(PropertyKeyConst.AccessKey, "{accessKey}");
consumerProperties.setProperty(PropertyKeyConst.SecretKey, "{secretKey}");
consumerProperties.setProperty(PropertyKeyConst.NAMESRV_ADDR, "{NAMESRV_ADDR}");
this.consumer = ExactlyOnceONSFactory.createExactlyOnceConsumer(consumerProperties);
this.consumer.subscribe("{topic}", "", new TestExactlyOnceListener());
consumer.start();
System.out.println("Consumer start success.") ;
}
}

/**
* SimpleListener is an example of the use of Exactly-Once delivery consumer for message consumption.
* It completes a simple process of recording messages in the database, and ensures that each message is
persistently stored in the database and takes effect only once.
*/
public class SimpleListener implements MessageListener {
private MQDataSource dataSource;

public SimpleListener() {
this.dataSource = new MQDataSource("{url}", "{user}", "{passwd}", "{driver}");
}

@Override
public Action consume(Message message, ConsumeContext context) {
Connection connection = null;
PreparedStatement statement = null;
try {
/**
* The consumed messages are processed for business accounting, and the processing results are persistently stored
in the database system by using MQDataSource.
* This example demonstrates how messages are consumed and recorded in the database system. The actual
business processing process is as follows: receive a message > process the message > persistently store the result.
* Exactly-Once delivery semantics ensures that each message is persistently stored only once.
*/
connection = dataSource.getConnection();
statement = connection.prepareStatement("INSERT INTO app(msg, ctime) VALUES(?, ?") ;
statement.setString(1, new String(message.getBody()));
statement.setLong(2, System.currentTimeMillis());
statement.execute();
System.out.println("consume message success");
return Action.CommitMessage;
} catch (Throwable e) {
System.out.println("consume message fail:" + e.getMessage());
return Action.ReconsumeLater;
} finally {
if (statement ! = null) {
try {
statement.close();
} catch (Exception e) {
}
}
if (connection ! = null) {
try {
connection.close();
} catch (Exception e) {
}
}
}
}
}
```

Create transactions on the message listener for database operations and

message consumption

Example:

```
/**
* Implement TestExactlyOnceListener
* Multiple business tables are updated in one transaction, and each operation in the transaction takes effect only once.
*/
public class SimpleTxListener implements MessageListener {
private MQDataSource dataSource;

public SimpleTxListener() {
this.dataSource = new MQDataSource("{url}", "{user}", "{passwd}", "{driver}");
}

@Override
public Action consume(Message message, ConsumeContext context) {
Connection connection = null;
Statement statement = null;
try {
/**
* The consumed messages are processed for business accounting, and the processing results are persistently stored in the database system by using MQDataSource.
* This example demonstrates an update of multiple tables in one transaction. The use of Exactly-Once delivery semantics ensures that each operation is performed only once in the transaction.
* The business processing logic is designed based on the following process: receive a message > process the message > persistently store the result.
*/
connection = dataSource.getConnection();
connection.setAutoCommit(false);
String insertSql = String.format("INSERT INTO app(msg, message_id, ctime) VALUES(\"%s\", \"%s\", %d)",
new String(message.getBody()), message.getMsgID(), System.currentTimeMillis());
String updateSql = String.format("UPDATE consume_count SET cnt = count + 1 WHERE consumer_group = \"%s\"",
"GID_TEST");
statement = connection.createStatement();
statement.execute(insertSql);
statement.execute(updateSql);
connection.commit();
System.out.println("consume message :" + message.getMsgID());
return Action.CommitMessage;
} catch (Throwable e) {
try {
connection.rollback();
} catch (Exception e1) {
}
System.out.println("consume message fail");
return Action.ReconsumeLater;
} finally {
if (statement ! = null) {
try {
statement.close();
} catch (Exception e) { }
}
if (connection ! = null) {
```

```
try {
connection.close();
} catch (Exception e) { }
}
}
}
}
```

Use the Spring Boot annotation to enable the Exactly-Once delivery semantics on the message listener

1. Create a message listener.

```


/**
* On the message listener, use an annotation to enable the Exactly-Once delivery semantics.
* You only need to add @MQTransaction to the consume method of the message listener.
* This method is applicable to the Spring Boot microservice.
*/

public class TestMessageListener implements MessageListener {
private final static String INSERTSQLFORMAT = "INSERT INTO app(message_id, ctime) VALUES(\"%s\", %d)";
private MQDataSource dataSource;

@Override
@MQTransaction
public Action consume(Message message, ConsumeContext context) {

Connection connection = null;
Statement statement = null;
try {
String insertSql = String.format(INSERTSQLFORMAT, message.getMsgID(), System.currentTimeMillis());
connection = this.dataSource.getConnection();
statement = connection.createStatement();
statement.execute(insertSql);
return Action.CommitMessage;
} catch (Throwable e) {
return Action.ReconsumeLater;
} finally {
if (statement ! = null) {
try {
statement.close();
} catch(Exception e) { }
}
if (connection ! = null) {
try {
connection.close();
} catch (Exception e) { }
}
}
}

public void setDataSource(MQDataSource dataSource) {
this.dataSource = dataSource;
}
}
```

1. Define a consumer bean and other related information in consumer.xml.

```
<? xml version="1.0" encoding="UTF-8"? >
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="mqDataSource"
class="com.aliyun.openservices.ons.api.impl.MQ.exactlyonce.datasource.MQDataSource" init-method="init"
destroy-method="close">
<property name="url" value="{url}" />
<property name="username" value="{user}" />
<property name="password" value="{passwd}" />
<property name="driverClass" value="{driver}" />
</bean>

<bean id="msgListener" class="com.aliyun.openservices.ons.api.impl.MQ.exactlyonce.spring.TestMessageListener">
<property name="dataSource" ref="mqDataSource"> <! --Consumer configuration-->
</property>
</bean> <! --Listener configuration -->

<! --When multiple group IDs subscribe to the same topic, you can create multiple consumer beans.
<bean id="consumer" class="com.aliyun.openservices.ons.api.bean.ExactlyOnceConsumerBean" init-
method="start" destroy-method="shutdown">
<property name="properties" > <! --Consumer configuration-->
<props>
<prop key="GROUP_ID">{gid}</prop>
<prop key="AccessKey">{accessKey}</prop>
<prop key="SecretKey">{secretKey}</prop>
<! --Set the number of consumer threads to 50.
<prop key="ConsumeThreadNums">50</prop>
-->
</props>
</property>
<property name="subscriptionTable">
<map>
<entry value-ref="msgListener">
<key>
<bean class="com.aliyun.openservices.ons.api.bean.Subscription">
<property name="topic" value="{topic}"/>
<property name="expression" value="{subExpression}"/><! --The expression is a tag. You can set it to a specific
tag, such as taga||tagb||tagc, or enter an asterisk (*) in this field. The asterisk (*) indicates subscription of all tags, but
is not used as a wildcard.-->
</bean>
</key>
</entry>
</map>
</property>
</bean>
</beans>
```

1. Run the consumer that is integrated with Spring.

```
public class ConsumeWithSpring {
public static void main(String[] args) {
/**
* The consumer bean is configured in consumer.xml. You can call ApplicationContext to obtain the bean or to inject
it to other classes, such as a controller.
*/
ApplicationContext context = new ClassPathXmlApplicationContext("consumer.xml");
System.out.println("Consumer Started");
}
```

Use MyBatis to enable the Exactly-Once delivery semantics on the message listener

1. Design the data access object (DAO) for write operations in the business database.

```
package com.aliyun.openservices.tcp.example.mybatis;

public interface AppDAO {
Integer insertMessage(String msgId);
}
```

1. Write the INSERT statement in the mapper.xml file.

```
<? xml version="1.0" encoding="UTF-8" ? >
<! DOCTYPE mapper
PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
"http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.aliyun.openservices.tcp.example.mybatis.AppDAO">
<insert id="insertMessage">
INSERT INTO app (message_id, ctime) VALUES (#{msgId}, now())
</insert>
</mapper>
```

1. Call MQDataSource to define the custom DataSourceFactory.

```
public class MQDataSourceFactoty extends DruidDataSourceFactory implements DataSourceFactory {
protected Properties properties;

@Override
public void setProperties(Properties properties) {
this.properties = properties;
}

@Override
public DataSource getDataSource() {
try {
DruidDataSource druidDataSource = (DruidDataSource) createDataSource(this.properties);
return new MQDataSource(druidDataSource);
} catch (Exception e) {
System.out.println("err:" + e.getMessage());
}
return null;
}
}
```

1. Register the data source as the MQDataSourceFactoty class in mybatis-config.xml.

```
<configuration>
<environments default="development">
<environment id="development">
<transactionManager type="JDBC"/>
<! --Database connection configuration-->
<dataSource type="com.aliyun.openservices.tcp.example.mybatis.MQDataSourceFactoty">
<property name="driverClass" value="com.mysql.jdbc.Driver"/>
<property name="url" value="{url}"/>
<property name="username" value="{username}"/>
<property name="password" value="{password}"/>

<property name="initialSize" value="10" />
<property name="maxActive" value="20"/>
</dataSource>
</environment>
</environments>
<mappers>
<mapper resource="mapper.xml"/>
</mappers>
</configuration>
```

1. Connect to the database from the message listener by using MyBatis to implement Exactly-Once consumption.

```
public class TestMybatisListener implements MessageListener {
private static SqlSessionFactory sqlSessionFactory;

static {
String resource = "mybatis-config.xml";
Reader reader = null;
try {
reader= Resources.getResourceAsReader(resource);
} catch (IOException e) {
e.printStackTrace();
}
sqlSessionFactory = new SqlSessionFactoryBuilder().build(reader);
}

@Override
public Action consume(Message message, ConsumeContext context) {
long begion = System.currentTimeMillis();
SqlSession sqlSession = null;
try {
sqlSession = sqlSessionFactory.openSession();
AppDAO appDAO = sqlSession.getMapper(AppDAO.class);
appDAO.insertMessage(message.getMsgID());
System.out.println("consume : " + message.getMsgID());
sqlSession.commit();
return Action.CommitMessage;
} catch (Exception e) {
e.printStackTrace();
sqlSession.rollback();
return Action.ReconsumeLater;
} finally {
sqlSession.close();
}
}
}
```

Precautions

When using an Exactly-Once consumer of MQ for message consumption, pay attention to the following points:

- The consumer offset cannot be reset manually in the console. If you reset the consumer offset to a consumed time point, the Exactly-Once delivery semantics becomes ineffective.

- Each insert or update operation in a database triggers an update operation. In addition, the Exactly-Once consumer queries and deletes data in the database periodically. These extra operations increase the IOPS of the database.

Send normal messages (in three modes)

MQ can send normal messages in three modes: reliable synchronous mode, reliable asynchronous mode, and one-way mode. This topic describes the working principles, scenarios, and differences of the three modes, and provides sample code for reference.

Note: Ordered messages can only be sent in reliable synchronous mode.

- **Reliable synchronous transmission**

Principle: In synchronous mode, the sender waits for the response from the receiver to the last data message before sending the next data message.

Scenario: This mode is applicable to extensive scenarios, such as email notification delivery, registration SMS notification delivery, and marketing SMS delivery.

- **Reliable asynchronous transmission**

Principle: In asynchronous mode, the sender sends the next data message without waiting for the receiver’ s response to the last data message. MQ needs to call the SendCallback operation to implement asynchronous transmission. The sender sends the second message immediately after the first one, without waiting for the response from the broker. The sender calls the SendCallback operation to receive responses from the broker and processes the responses.

Scenario: This mode is used on time-consuming links for business scenarios that are sensitive to response time (RT). For example, after a user uploads a video, a notification is sent to enable the encoding service. After encoding is complete, a notification is sent to return the encoding result.

- **One-way transmission**

Principle: In one-way transmission mode, the sender only sends messages and does not wait for broker responses or call any callback function. This mode consumes the least time and completes transmission within microseconds.

Scenario: This mode is applicable to least time-consuming scenarios that pose low reliability requirements, for example, log collection.

The following table summarizes the features of the three transmission modes and the differences between them.

Transmission mode	TPS	Response	Reliability
Synchronous transmission	High	Yes	No message loss
Asynchronous transmission	High	Yes	No message loss
One-way transmission	Highest	None	Possible message loss

Sample code

Synchronous transmission

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import java.util.Properties;

public class ProducerTest {
public static void main(String[] args) {
Properties properties = new Properties();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey,"XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the message transmission timeout period (in milliseconds).
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");

Producer producer = ONSFactory.createProducer(properties);
//Before sending a message, call the start() method once to start the producer.
producer.start();

// Send messages cyclically.
for (int i = 0; i < 100; i++){
Message msg = new Message( //
// The topic of the message.
"TopicTestMQ",
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter messages on the MQ broker based on the specified criteria.
"TagA",
// The message body in any binary format. MQ does not process the message body.
// The producer and consumer must negotiate the consistent serialization and deserialization methods.
"Hello MQ".getBytes());
// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the Alibaba Cloud console if you fail to receive the message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_" + i);

try {
SendResult sendResult = producer.send(msg);
// Send the message in synchronous mode. The message is sent if no exception is thrown.
if (sendResult ! = null) {
System.out.println(new Date() + " Send mq message success. Topic is:" + msg.getTopic() + " msgId is: " + sendResult.getMessageId());
}
}
catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getTopic());
e.printStackTrace();
}
}

// Destroy the producer object before exiting from the application.
// Note: You can choose not to destroy the producer object.
producer.shutdown();
}
}
```

Asynchronous transmission

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.OnExceptionContext;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.SendCallback;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import java.util.Properties;

public static void main(String[] args) {
Properties properties = new Properties();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the message transmission timeout period (in milliseconds).
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");

Producer producer = ONSFactory.createProducer(properties);
//Before sending a message, call the start() method once to start the producer.
producer.start();

Message msg = new Message(
// The topic of the message.
"TopicTestMQ",
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter
messages on the MQ broker based on the specified criteria.
"TagA",
// The message body in any binary format. MQ does not process the message body. The producer and consumer
must negotiate the consistent serialization and deserialization methods.
"Hello MQ".getBytes());

// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
A unique identifier enables you to query a message and resend it in the console if you fail to receive the message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_100");

// Send the message in asynchronous mode. The result is returned to the client through the callback function.
producer.sendAsync(msg, new SendCallback() {
@Override
public void onSuccess(final SendResult sendResult) {
// The message is sent to the consumer.
System.out.println("send message success. topic=" + sendResult.getTopic() + ", msgId=" +
sendResult.getMessageId());
}

@Override
public void onException(OnExceptionContext context) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data
persistently.
System.out.println("send message failed. topic=" + context.getTopic() + ", msgId=" + context.getMessageId());
}
});

// The message ID can be obtained before the callback function returns the result.
System.out.println("send message async. topic=" + msg.getTopic() + ", msgId=" + msg.getMsgID());

// Destroy the producer object before exiting from the application. Note: You can choose not to destroy the
producer object.
producer.shutdown();
}
```

One-way transmission

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import java.util.Properties;

public static void main(String[] args) {
Properties properties = new Properties();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the message transmission timeout period (in milliseconds).
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");

Producer producer = ONSFactory.createProducer(properties);
//Before sending a message, call the start() method once to start the producer.
producer.start();
// Send messages cyclically.
for (int i = 0; i < 100; i++){
Message msg = new Message(
// The topic of the message.
"TopicTestMQ",
// The message tag,
// It is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter messages on the MQ
broker based on the specified criteria.
"TagA",
// The message body
// It is in any binary format. MQ does not process the message body. The producer and consumer must negotiate
the consistent serialization and deserialization methods.
"Hello MQ".getBytes());

// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the Alibaba Cloud console if you fail to
receive the message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_" + i);
// In one-way transmission mode, the sender does not wait for the response from the broker. Therefore, if
messages that fail to be delivered are not retransmitted, data is lost. If data loss is not acceptable, we recommend
that you use the reliable synchronous or asynchronous transmission mode.
producer.sendOneway(msg);
}

// Destroy the producer object before exiting from the application.
// Note: You can choose not to destroy the producer object.
producer.shutdown();
}
```

Send messages (using multiple threads)

The consumer and producer client objects of MQ are thread-secure and can be shared among multiple threads.

You can deploy multiple producer and consumer instances on one or more servers. A producer or consumer instance can also run multiple threads to send or receive messages, improving the message transmitting or receiving TPS. Do not create a client instance for every thread.

The sample code for sharing a producer instance among multiple threads is as follows:

```

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import com.aliyun.openservices.ons.api.SendResult;
import java.util.Properties;

public class SharedProducer {
public static void main(String[] args) {
// Initialize the producer configuration.
Properties properties = new Properties();
// The group ID you created in the console.
properties.put(PropertyKeyConst.GROUP_ID, "XXX");
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey,"XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the message transmission timeout period (in milliseconds).
properties.setProperty(PropertyKeyConst.SendMsgTimeoutMillis, "3000");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");
final Producer producer = ONSFactory.createProducer(properties);
//Before sending a message, call the start() method once to start the producer.
```

```
producer.start();

// The created producer and consumer objects are thread-secure and can be shared among multiple threads. Do
not create a client instance for every thread.

// Two threads share the producer object and concurrently send messages to MQ.
Thread thread = new Thread(new Runnable() {
@Override
public void run() {
try {
Message msg = new Message( //
// The topic of the message.
"TopicTestMQ",
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter
messages on the MQ broker based on the specified criteria.
"TagA",
// The message body in any binary format. MQ does not process the message body.
// The producer and consumer must negotiate the consistent serialization and deserialization methods.
"Hello MQ".getBytes());
SendResult sendResult = producer.send(msg);
// Send the message in synchronous mode. The message is sent if no exception is thrown.
if (sendResult ! = null) {
System.out.println(new Date() + " Send mq message success. Topic is:" + MqConfig.TOPIC + " msgId is: " +
sendResult.getMessageId());
}
} catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data
persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + MqConfig.TOPIC);
e.printStackTrace();
}
}
});
thread.start();


Thread anotherThread = new Thread(new Runnable() {
@Override
public void run() {
try {
Message msg = new Message("TopicTestMQ", "TagA", "Hello MQ".getBytes());
SendResult sendResult = producer.send(msg);
// Send the message in synchronous mode. The message is sent if no exception is thrown.
if (sendResult ! = null) {
System.out.println(new Date() + " Send mq message success. Topic is:" + MqConfig.TOPIC + " msgId is: " +
sendResult.getMessageId());
}
} catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data
persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + MqConfig.TOPIC);
e.printStackTrace();
}
}
});
anotherThread.start();
```

```
// If the producer instance is no longer used, disable it to release resources.
// producer.shutdown();
}
}
```

Send and receive ordered messages

Use Java SDK 1.2.7 or later to send and subscribe to ordered messages.

Ordered messages are delivered and consumed in order. MQ provides this type of messages for applications that require message delivery and consumption in strict FIFO order. For more information, see Ordered messages.

Globally and partitionally ordered messages are sent and received in similar ways. See the following sample code for reference.

Send ordered messages

The sample code for sending messages is as follows:

```
package com.aliyun.openservices.ons.example.order;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.order.OrderProducer;

import java.util.Properties;


public class ProducerClient {

public static void main(String[] args) {
Properties properties = new Properties();
// The group ID you created in the console.
properties.put(PropertyKeyConst.GROUP_ID, "XXX");
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");
OrderProducer producer = ONSFactory.createOrderProducer(properties);
//Before sending a message, call the start() method once to start the producer.
producer.start();
for (int i = 0; i < 1000; i++) {
String orderId = "biz_" + i % 10;
Message msg = new Message(//
// The topic of the message.
"Order_global_topic ",
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter
messages on the MQ broker based on the specified criteria.
"TagA",
// The message body in any binary format. MQ does not process the message body. The producer and consumer
must negotiate the consistent serialization and deserialization methods.
"send order global msg".getBytes()
);
// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the console if you fail to receive the
message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey(orderId);
// The key field that identifies the partition of partitionally ordered messages. This sharding key is different from the
key of normal messages.
// This field can be set to any non-empty string for globally ordered messages.
String shardingKey = String.valueOf(orderId);
try {
SendResult sendResult = producer.send(msg, shardingKey);
// The message is sent if no exception is thrown.
if (sendResult ! = null) {
System.out.println(new Date() + " Send mq message success. Topic is:" + msg.getTopic() + " msgId is: " +
sendResult.getMessageId());
}
}
catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data
persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getTopic());
e.printStackTrace();
}
}
// Destroy the producer object before exiting from the application.
// Note: You can choose not to destroy the producer object.
producer.shutdown();
}

}
```

Subscribe to ordered messages

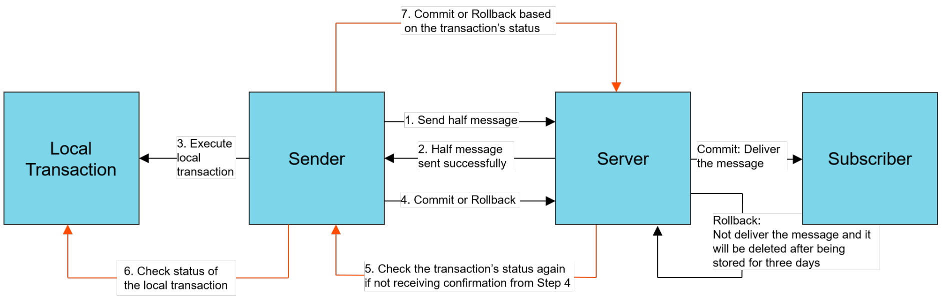
The sample code for subscribing to ordered messages is as follows:

```
package com.aliyun.openservices.ons.example.order;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.order.ConsumeOrderContext;
import com.aliyun.openservices.ons.api.order.MessageOrderListener;
import com.aliyun.openservices.ons.api.order.OrderAction;
import com.aliyun.openservices.ons.api.order.OrderConsumer;

import java.util.Properties;


public class ConsumerClient {

public static void main(String[] args) {
Properties properties = new Properties();
// The group ID you created in the console.
properties.put(PropertyKeyConst.GROUP_ID, "XXX");
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");
// Set the delay time (in milliseconds) before the retry upon an ordered message consumption failure. Value range:
10~1800.
properties.put(PropertyKeyConst.SuspendTimeMillis, "100");
// Set the maximum number of retries upon a message consumption failure.
properties.put(PropertyKeyConst.MaxReconsumeTimes,"20");

// Before message subscription, call the start() method once to start the consumer.
OrderConsumer consumer = ONSFactory.createOrderedConsumer(properties);

consumer.subscribe(
// The topic of the message.
"Jodie_Order_Topic ",
// Subscribe to message tags under the specified topic.
// 1. * indicates the subscription of all messages.
// 2. TagA || TagB || TagC indicates the subscription of messages with TagA, TagB, or TagC.
"*",
new MessageOrderListener() {
/**
* 1. OrderAction.Suspend is returned if a message fails to be consumed or an exception occurs during message
processing.<br>
* 2. OrderAction.Success is returned if a message is processed.
*/
@Override
public OrderAction consume(Message message, ConsumeOrderContext context) {
System.out.println(message);
return OrderAction.Success;
}
});

consumer.start();
}
}
```

Send and receive transactional messages

Interaction process

The following figure shows the transactional message interaction among MQ components.


7. Commit or Rollback based on the transaction's status
Local Transaction
3. Execute local transaction
Sender
1. Send half message
2. Half message sent successfully
4. Commit or Rollback
Server
Commit: Deliver the message
Subscriber
Rollback: Not deliver the message and it will be deleted after being stored for three days
6. Check status of the local transaction
5. Check the transaction's status again if not receiving confirmation from Step 4

Send transactional messages

Follow these steps to send a transactional message:

1. Send a half message and execute a local transaction. The sample code is as follows:

```
package com.alibaba.webx.TryHsf.app1;

import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.PropertyKeyConst;
import com.aliyun.openservices.ons.api.SendResult;
import com.aliyun.openservices.ons.api.transaction.LocalTransactionExecuter;
import com.aliyun.openservices.ons.api.transaction.TransactionProducer;
import com.aliyun.openservices.ons.api.transaction.TransactionStatus;
import java.util.Properties;
import java.util.concurrent.TimeUnit;

public class TransactionProducerClient {
private final static Logger log = ClientLogger.getLog(); // Set your own logger to facilitate troubleshooting.

public static void main(String[] args) throws InterruptedException {
final BusinessService businessService = new BusinessService(); // Local business
Properties properties = new Properties();
// The group ID you created in the console. Note: Transactional messages cannot share the group ID with other
types of messages.
properties.put(PropertyKeyConst.GROUP_ID, "XXX");
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");

TransactionProducer producer = ONSFactory.createTransactionProducer(properties,
new LocalTransactionCheckerImpl());
producer.start();
Message msg = new Message("Topic", "TagA", "Hello MQ transaction===".getBytes());
try {
SendResult sendResult = producer.send(msg, new LocalTransactionExecuter() {
@Override
public TransactionStatus execute(Message msg, Object arg) {
// The message ID. Two messages may have the same message body but different message IDs. The current message ID cannot be obtained in the console.
String msgId = msg.getMsgID();
// Compute the message body by using CRC32, MD5, or other algorithms.
long crc32Id = HashUtil.crc32Code(msg.getBody());
// The message ID and CRC32 ID are used to prevent duplication of messages.
// You do not need to specify the message ID or CRC32 ID if the business itself achieves idempotence. Otherwise, set the message ID or CRC32 ID to ensure idempotence.
// To avoid duplication of messages, compute the message body by using the CRC32 or MD5 algorithm.
Object businessServiceArgs = new Object();
TransactionStatus transactionStatus = TransactionStatus.Unknow;
try {
boolean isCommit =
businessService.execbusinessService(businessServiceArgs);
if (isCommit) {
// Submit the message if the local transaction succeeds.
transactionStatus = TransactionStatus.CommitTransaction;
} else {
// Roll back the message if the local transaction fails.
transactionStatus = TransactionStatus.RollbackTransaction;
}
} catch (Exception e) {
log.error("Message Id:{}", msgId, e);
}
System.out.println(msg.getMsgID());
log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
return transactionStatus;
}
}, null);
}
catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getTopic());
e.printStackTrace();
}
// The demo example to prevent the process from exiting (not necessary in practical use)
TimeUnit.MILLISECONDS.sleep(Integer.MAX_VALUE);
}
}
```

1. Submit the transactional message status.

After the execution of a local transaction (successful or failed), the broker must be notified of the transaction status of the current message. Two notification modes are supported:

- Submit the status after executing the local transaction.
- Wait until the broker requests to check the transaction status of the message.

A transaction may be in one of the following states:

- TransactionStatus.CommitTransaction: The transaction is submitted, and the consumer can consume the message.
- TransactionStatus.RollbackTransaction: The transaction is rolled back, and the message is discarded and cannot be consumed.
- TransactionStatus.Unknown: The transaction status is unknown, and the sender is waiting for the MQ broker to query the transaction status of the message.

```
public class LocalTransactionCheckerImpl implements LocalTransactionChecker {
private final static Logger log = ClientLogger.getLog();
final BusinessService businessService = new BusinessService();

@Override
public TransactionStatus check(Message msg) {
// The message ID (Two messages may have the same message body but different message IDs. The current
message is a half message, and therefore its message ID cannot be obtained in the console.)
String msgId = msg.getMsgID();
// Compute the message body by using CRC32, MD5, or other algorithms.
long crc32Id = HashUtil.crc32Code(msg.getBody());
// The message ID and CRC32 ID are used to prevent duplication of messages.
// You do not need to specify the message ID or CRC32 ID if the business itself achieves idempotence. Otherwise,
set the message ID or CRC32 ID to ensure idempotence.
// To eliminate the duplication of messages, we recommend that you use CRC32 or MD5 to process the message
body.
// The parameter object of the business. This is an example. Set the object based on the actual situation of your
business.
Object businessServiceArgs = new Object();
TransactionStatus transactionStatus = TransactionStatus.Unknow;
try {
boolean isCommit = businessService.checkbusinessService(businessServiceArgs);
if (isCommit) {
// Submit the message if the local transaction succeeds.
transactionStatus = TransactionStatus.CommitTransaction;
} else {
// Roll back the message if the local transaction fails.
transactionStatus = TransactionStatus.RollbackTransaction;
}
} catch (Exception e) {
log.error("Message Id:{}", msgId, e);
}
log.warn("Message Id:{}transactionStatus:{}", msgId, transactionStatus.name());
return transactionStatus;
}
}
```

Tool class

```


import java.util.zip.CRC32;
public class HashUtil {
public static long crc32Code(byte[] bytes) {
CRC32 crc32 = new CRC32();
crc32.update(bytes);
return crc32.getValue();
}
}
```

Transaction check mechanism

- Why is the check mechanism required for transactional message delivery?

If the half message is sent in step 1 but TransactionStatus.Unknown is returned, or if no status is submitted because the application exits, the status of the half message is unknown to the broker. Therefore, the broker periodically requests the sender to check and report the status of the half message.

- What does the business logic do when the check method is called back?

The check method for transactional messages needs to contain the logic of transaction consistency check. After a transactional message is sent, MQ needs to use the LocalTransactionChecker operation to respond to the request of the broker for the local transaction status. Therefore, the check method for the transactional message needs to complete the following tasks:

(1) Check the status of the local transaction corresponding to the half message (committed or rollback).

(2) Submit the status of the local transaction to the broker.

Subscribe to transactional messages

The method for subscribing to transactional messages is the same as that for subscribing to normal messages. For more information, see Subscribe to messages.

Send and receive delayed messages

Delayed messages are delivered to a consumer after a delay (3 seconds, for example) from when they are sent to the MQ broker. Send this type of message when a time window is required between the production and consumption of the message, or when tasks need to be triggered after a delay. Delayed messages are similar to delayed queues.

For more information about the concept behind delayed messages and for precautions for use of these messages, see Delayed messages.

Send delayed messages

The sample code for sending delayed messages is as follows:

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import com.aliyun.openservices.ons.api.SendResult;
import java.util.Properties;

public class ProducerDelayTest {
public static void main(String[] args) {
Properties properties = new Properties();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");

Producer producer = ONSFactory.createProducer(properties);
//Before sending a message, call the start() method once to start the producer.
producer.start();
Message msg = new Message( //
// The topic you created in the console.
"Topic",
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter messages on the MQ broker based on the specified criteria.
"tag",
// The message body in any binary format. MQ does not process the message body. The producer and consumer must negotiate the consistent serialization and deserialization methods.
"Hello MQ".getBytes());
// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the console if you fail to receive the
message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_100");
try {
// The message delivery delay time, in milliseconds (ms). Messages are delivered after the specified delay (against the current time), for example, 3 seconds.
long delayTime = System.currentTimeMillis() + 3000;

// Set the message delivery time.
msg.setStartDeliverTime(delayTime);

SendResult sendResult = producer.send(msg);
// Send the message in synchronous mode. The message is sent if no exception is thrown.
if (sendResult ! = null) {
System.out.println(new Date() + " Send mq message success. Topic is:" + msg.getTopic() + " msgId is: " + sendResult.getMessageId());
}
} catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getTopic());
e.printStackTrace();
}
// Destroy the producer object before exiting from the application.<br>
// Note: You can choose not to destroy the producer object.
producer.shutdown();
}
}
```

Subscribe to delayed messages

The method for subscribing to delayed messages is the same as that for subscribing to normal messages. For more information, see Subscribe to messages.

Send and receive scheduled messages

Scheduled messages are consumed after a specified timestamp. These messages are sent when a time window is required between message production and consumption, or when tasks need to be triggered at a scheduled time.

For information about the concept of scheduled messages and use precautions for these messages, see Scheduled messages.

Send scheduled messages

The sample code for sending scheduled messages is as follows:

```
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.Producer;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import com.aliyun.openservices.ons.api.SendResult;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Properties;

public class ProducerDelayTest {
public static void main(String[] args) {
Properties properties = new Properties();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");
Producer producer = ONSFactory.createProducer(properties);
//Before sending a message, call the start() method once to start the producer.
producer.start();
Message msg = new Message( //
// The topic of the message.
"Topic",
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter messages on the MQ broker based on the specified criteria.
"tag",
// The message body in any binary format. MQ does not process the message body. The producer and consumer must negotiate the consistent serialization and deserialization methods.
"Hello MQ".getBytes())
// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the console if you fail to receive the message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_100");

try {
// The scheduled message delivery time (unit: ms) after a specific timestamp, for example, 2016-03-07 16:21:00. If the scheduled time is earlier than the current timestamp, the message is immediately delivered to the consumer.
long timeStamp = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss").parse("2016-03-07 16:21:00").getTime();

msg.setStartDeliverTime(timeStamp);
// The message is sent if no exception is thrown.
SendResult sendResult = producer.send(msg);
System.out.println("Message Id:" + sendResult.getMessageId());
}
catch (Exception e) {
// The message failed to be sent and requires a retry. The system can resend the message or store message data persistently.
System.out.println(new Date() + " Send mq message failed. Topic is:" + msg.getTopic());
e.printStackTrace();
}

// Destroy the producer object before exiting from the application.
// Note: You can choose not to destroy the producer object.
producer.shutdown();
}
}
```

Subscribe to scheduled messages

The method for subscribing to scheduled messages is the same as that for subscribing to normal messages. For more information, see Subscribe to messages.

Subscribe to messages

This topic describes how to subscribe to messages by using the Java SDK of MQ.

Note:

- Maintain consistent subscription for all consumer instances with the same group ID. For more information, see Subscription consistency.

Subscription modes

MQ supports the following two message subscription modes:

Clustering subscription: All the consumers identified by the same group ID equally share messages. For example, a topic contains nine messages and a group contains three consumer instances. In this case, each instance consumes three messages.

```
 // The configuration of clustering subscription (default mode).
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
```

Broadcasting subscription: All the consumers identified by the same group ID consume every message once. For example, a topic contains nine messages and a group contains three consumer instances. In this case, each instance consumes nine messages.

```
 // The configuration of broadcasting subscription.
properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);
```

Sample code

```
import com.aliyun.openservices.ons.api.Action;
import com.aliyun.openservices.ons.api.ConsumeContext;
import com.aliyun.openservices.ons.api.Consumer;
import com.aliyun.openservices.ons.api.Message;
import com.aliyun.openservices.ons.api.MessageListener;
import com.aliyun.openservices.ons.api.ONSFactory;
import com.aliyun.openservices.ons.api.PropertyKeyConst;

import java.util.Properties;

public class ConsumerTest {
public static void main(String[] args) {
Properties properties = new Properties();
// The group ID you created in the console.
properties.put(PropertyKeyConst.GROUP_ID, "XXX");
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.AccessKey, "XXX");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
properties.put(PropertyKeyConst.SecretKey, "XXX");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
properties.put(PropertyKeyConst.NAMESRV_ADDR,
"XXX ");
// Clustering subscription (default)
// properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.CLUSTERING);
// Broadcasting subscription
// properties.put(PropertyKeyConst.MessageModel, PropertyValueConst.BROADCASTING);

Consumer consumer = ONSFactory.createConsumer(properties);
consumer.subscribe("TopicTestMQ", "TagA||TagB", new MessageListener() { // Subscribe to multiple tags.
public Action consume(Message message, ConsumeContext context) {
System.out.println("Receive: " + message);
return Action.CommitMessage;
}
});

// Subscribe to another topic.
consumer.subscribe("TopicTestMQ-Other", "*", new MessageListener() { // Subscribe to all tags.
public Action consume(Message message, ConsumeContext context) {
System.out.println("Receive: " + message);
return Action.CommitMessage;
}
});

consumer.start();
System.out.println("Consumer Started");
}
}
```

Note:

In broadcasting consumption mode, you cannot set a message accumulation alarm or query the message accumulation condition in the console. Therefore, you can create multiple group IDs to enable message broadcasting. For more information, see **Use the clustering consumption mode to simulate the broadcasting consumption mode** in Clustering and broadcasting consumption.

For more information about the best practices for throttling on MQ consumer clients, see MQ client traffic control design.

C/C++ SDK

Release Notes

This topic provides the download links, versions, and updates of all C++ SDKs so that you can choose a suitable one for use.

ons-cpp v1.1.2

Release date	Version	Download (Windows version)	Download (Linux version)	Environment preparation guide
2019-01-16	1.1.2	aliyun-mq-windows-cpp-sdk.zip	aliyun-mq-linux-cpp-sdk.tar.gz	Prepare the C/C++ SDK environment

New features

Enabled instance user access to the service in either of the following modes (that for non-instance users unchanged):

Configure NAMESRV_ADDR with InstanceId.

Configure InstanceId and NAMESRV_ADDR without InstanceId.

Replaced ProducerId and ConsumerId with GroupId.

ons-cpp v1.1.1

Release date	Version	Download (Windows version)	Download (Linux version)	Environment preparation guide
2018-07-31	1.1.1	aliyun-mq-windows-cpp-sdk.zip	aliyun-mq-linux-cpp-sdk.tar.gz	Prepare the C/C++ SDK environment

New features

- Supported SSL encrypted transmission. (Note: This function is only applicable to the MQ Enterprise Platinum Edition.)
- Added the feature that PushConsumer pulls messages in asynchronous mode to improve the message push efficiency.

Bugs fixed

- Fixed issues related to ordered messages.
- Optimized logging so that logs are printed only when Rebalance results are changed.
- Fixed the problem that the system flag is not serialized to the one-way request header.

More historical versions

ons-cpp v1.1.0

Release date	Version	Download (Windows version)	Download (Linux version)	Environment preparation guide
2017-07-25	1.1.0	aliyun-mq-windows-cpp-sdk.zip	aliyun-mq-linux-cpp-sdk.zip	Prepare the C/C++ SDK environment

Bugs fixed

- Fixed coredump caused by consumer shutdown.
- Fixed the problem that the underlying URL class does not support HTTP access on Windows.
- Fixed the timestamp error of message trace.
- Fixed the problem that an incorrect IP address is displayed in message trace.
- Fixed the problem of memory leakage on Windows.

ons-cpp v1.0.9

Release date	Version	Download (Windows version)	Download (Linux version)	Environment preparation guide
2016-12-29	1.0.9	-	-	Prepare the C/C++ SDK environment

New features

- Supported the transmission of one-way messages.
- Added ordered messages.
- Added the timeout duration settings for ordered messages.
- Added the message retry count settings.

Bugs fixed

- Fixed the problem of resource leakage at shutdown.
- Fixed the problem of coredump at shutdown.

ons-cpp v1.0.8

Release date	Version	Download (Windows version)	Download (Linux version)	Environment preparation guide
2016-12-02	1.0.8	-	-	Prepare the C/C++ SDK environment

New features

- Abandoned the old C# SDK. Instead, a new C# SDK is generated with SWIG, which is more stable in the ASP.NET support.
- Supported log path customization.
- Provided built-in Chinese UTF-8 encoding. Users no longer need to explicit encoding and decoding.
- Added the MQ_GUIDE document and ASP.NET demo.

Function optimization

- Upgraded the boost library version to 1.6.2.

Bugs fixed

- Fixed the problem that coredump occurs when an ordered message exits.

ons-cpp v1.0.7

Release date	Version	Download (Windows version)	Download (Linux version)	Environment preparation guide
2016-11-15	1.0.7	-	-	Prepare the C/C++ SDK environment

New features

- Added consumption throttling for consumers. By default, 1,000 messages are pulled and stored in the memory. Then, callback functions of users are called back one by one.
- Added ordered messages.
- Added the timeout duration settings for ordered messages.
- Added the message retry count settings.

Function optimization

- Improved the message tracing function by sending trace data in a separate thread pool.
- Optimized the TCP lock granularity.

Bugs fixed

- Fixed several bugs in message tracing.
- Fixed the problem of coredump at shutdown.
- Fixed the problem of memory leakage.
- Fixed the problem that an exception is thrown when the message tag contains the special character "||".

Prepare the C_C++ SDK environment

The following preparation is required for accessing MQ through the C++ SDK.

Note:

You need to create the topic and group ID involved in the code in the MQ console first. The message tag can be specified by the application users. For more information about the creation process, see **Step 2: Create resources** in Quick start for primary accounts.

Applications that use MQ must be deployed on Alibaba Cloud ECS instances.

Download SDK

C++ supports cross-platform SDKs for both Windows and Linux, and the APIs are the same. For more information about the download URL of the latest C++ SDK, see Release Notes.

Download and decompress the .zip package of the C++ SDK. The .zip package contains the following directories and files:

```
- example/
- include/
- lib/
- SDK_GUIDE.pdf
- changelog
```

The preceding directories and files serve the following purposes:

demo: This folder contains a created Windows C++ demo.

example:This folder contains examples for sending and consuming normal messages and ordered messages and examples for sending messages in one-way mode. Besides, the package for Linux also contains a file Makefile for compiling and managing the examples.

include: This folder contains the header file to be included in your program.

lib: The Linux SDK sub-directories are as follows, which are 64-bit status library and dynamic library, respectively.

```
lib-boost-share/
libonsclient4cpp.so
lib-boost-static/
libonsclient4cpp.a
```

The Windows SDK sub-directories are as follows, which is the 64-bit SDK DLL library. If Visual Studio 2015 is not installed, install vc_redist.x64. This is the runtime environment for Visual C++ 2015.

```
64/
vc_redist.x64
```

SDK_GUIDE.pdf: This file describes how to prepare the SDK environment and contains FAQ.

changelog: This file lists the problems that have been fixed and the new features of the new version.

Linux C++ SDK

From December 2, 2016, Linux C++ SDK added dependency on high performance Boost libraries (v1.62.0), which reduces the CPU resource usage and enhances the efficiency. Currently, Linux C++ SDK depends on four libraries: boost_system, boost_thread, boost_chrono, and boost_filesystem. We provide solutions for both static library and dynamic library.

Static solution

The MQ library file is in the directory lib/lib-boost-static and the Boost libraries are statically linked to libonsclient4cpp.a. Service providers who have no dependent Boost libraries can directly choose the static library solution. In the static library solution, the Boost libraries have been linked to ibonsclient4cpp.a, and you just need to link to libonsclient4cpp.a when compiling the file. The code is as follows:

```
cd aliyun-mq-linux-cpp-sdk  //The path to which the downloaded SDK package is decompressed.
cd example //Go to the demo directory and enter the topic and key you created and other information in the demo file.
make static=1
```

Note: For completely static links, make sure that required static libraries such as libstdc++ and pthread have been installed on your machine. libstdc ++ that is installed by default does not contain static libraries. You need to run the yum or > apt-get command to install the static libraries. The following warning message may be returned when you use the preceding method:

```
warning: Using 'gethostbyaddr' in statically linked applications requires at runtime the shared libraries from the glibc version used for linking
```

The best practice is to avoid the use of completely static links, but to use statically linked lonsclient4cpp only and link to other libraries dynamically. The code is as follows:

```
g++ -ggdb -Wall -O3 -I../include    ../example/ProducerExampleForEx.cpp  -Wl,-static -lonsclient4cpp -L../lib/lib-boost-static/ -Wl,-Bdynamic  -lpthread -ldl -lrt -o ../example/ProducerExampleForEx
```

Additionally, since GCC 5.x introduces Dual ABI, you need to add **-D_GLIBCXX_USE_CXX11_ABI=0** when compiling the links.

Dynamic solution

The MQ library file is in the directory lib/lib-boost-share. When generating executable files, service providers need to link the Boost dynamic libraries to libonsclient4cpp.so. Since the service providers have depended on Boost libraries, in scenarios where the dynamic library solution is required, perform the following steps for the dependency of Boost libraries:

1. Download Boost 1.62.0:

boost 1.62.0

1. Decompress the Boost 1.62.0 package:

tar —bzip2 -xf /path/to/boost_1_62_0.tar.bz2

1. Install Boost 1.62.0:

1) cd path/to/boost_1_62_0

2) Configure Boost: ./bootstrap.sh

3) Compile Boost: ./b2 link=shared runtime-link=shared

4) Install Boost: ./b2 install

Run ldconfig -v|grep libboost. If the system makes output, the boost dynamic libraries are in the search paths of the dynamic libraries.

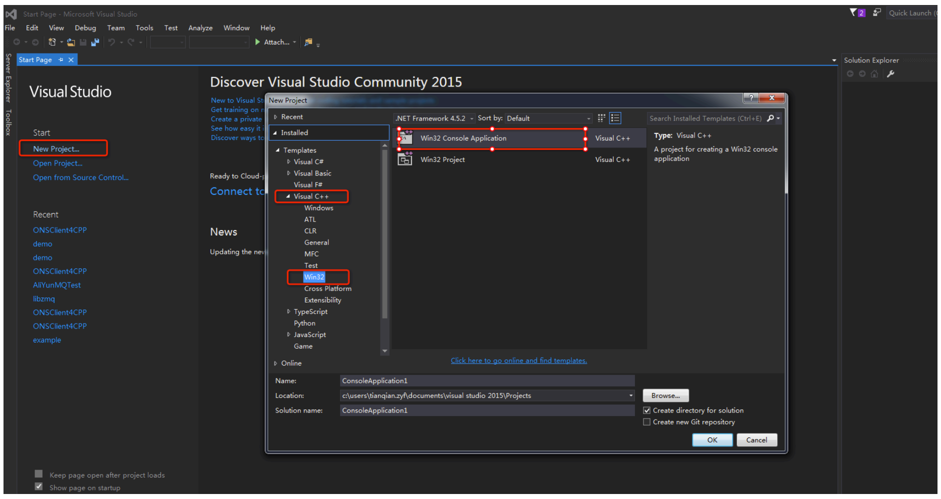
When executable files are generated, you need to link Boost dynamic libraries to the MQ dynamic library. The method for this is as follows:

```
cd aliyun-mq-linux-cpp-sdk  //The path to which the downloaded SDK package is decompressed.
cd example //Go to the demo directory and enter the topic and key you created in the MQ console and other information in the demo file.
g++ -Wall -Wno-deprecated -L ../lib/lib-boost-share/ -I ../include/ ProducerExampleForEx.cpp -lonsclient4cpp -lboost_system -lboost_thread -lboost_chrono -lboost_filesystem -lpthread
export LD_LIBRARY_PATH="../lib/lib-boost-share/" //Add dynamically loaded search paths.
./a.out //Run the program.
```

Windows C++ SDK

Use C++ SDK in the Visual Studio 2015 environment

Use Visual Studio 2015 to create your project.


New Project

Templates > Visual C++ > Win32

Win32 Console Application Visual C++

Type: Visual C++

A project for creating a Win32 console application

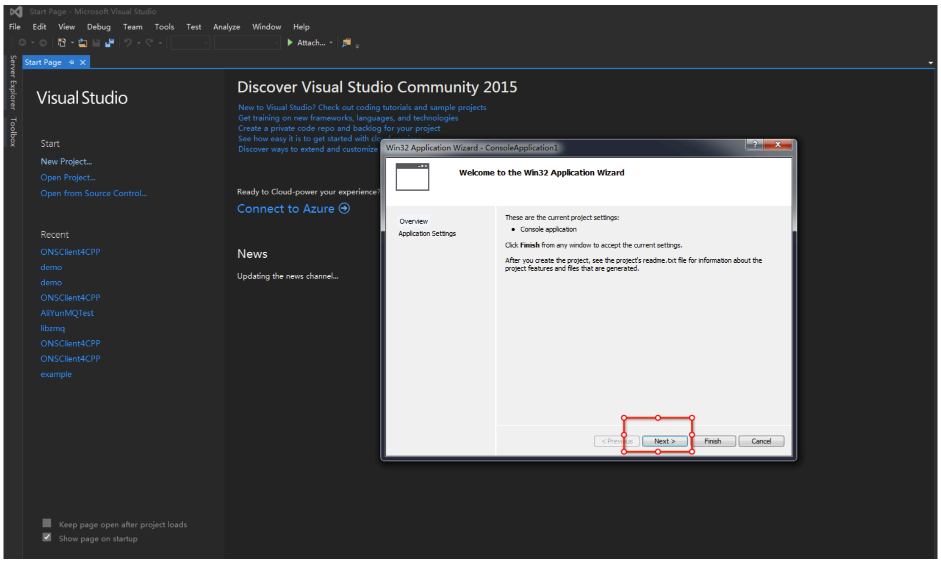
Name: ConsoleApplication1

Location: c:\users\tianqian.zyf\documents\visual studio 2015\Projects

Solution name: ConsoleApplication1

☑ Create directory for solution

☐ Create new Git repository


Win32 Application Wizard - ConsoleApplication1

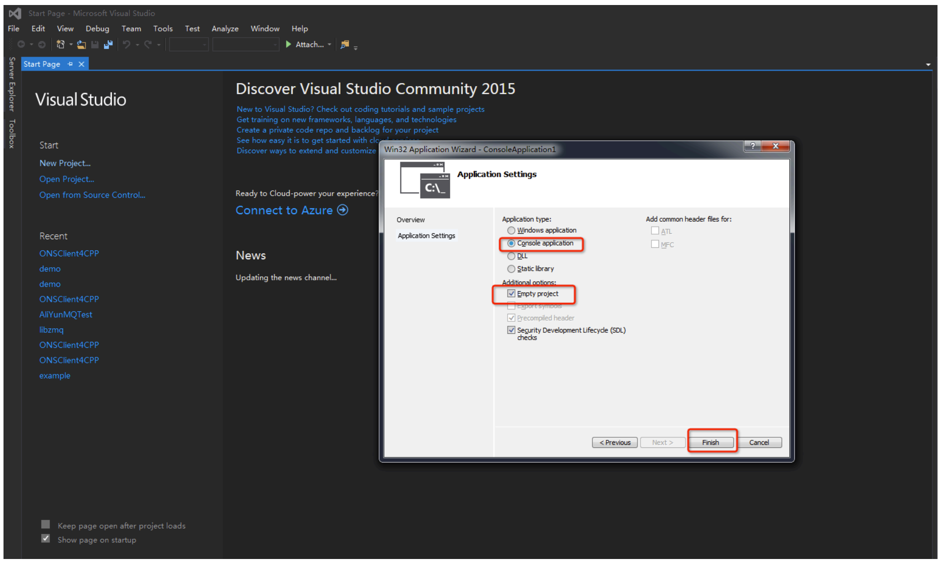
Welcome to the Win32 Application Wizard

These are the current project settings:

- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.


Win32 Application Wizard - ConsoleApplication1

Application Settings

Application type:

◯ Windows application

● Console application

◯ DLL

◯ Static library

Additional options:

☑ Empty project

☑ Precompiled header

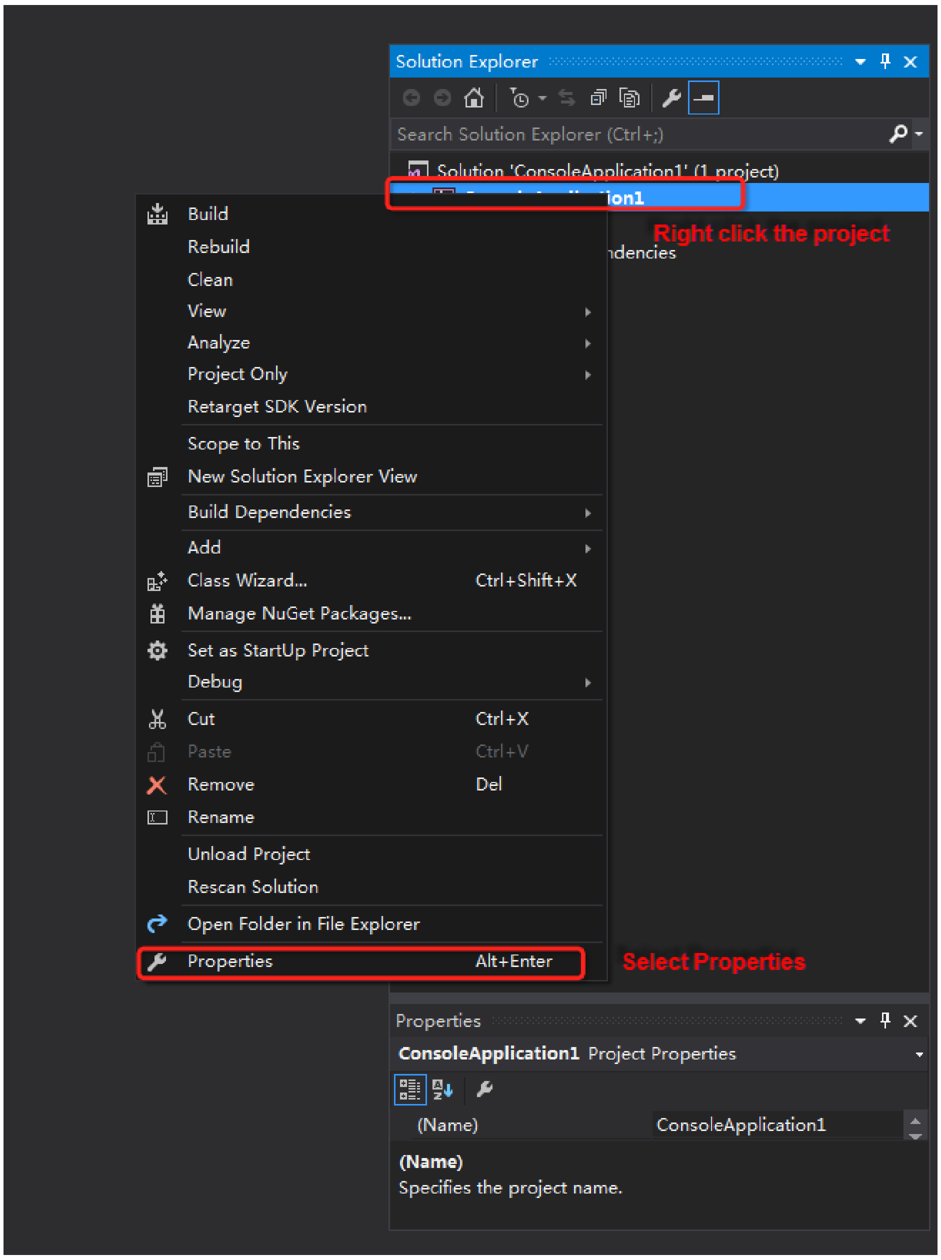
☑ Security Development Lifecycle (SDL) checks

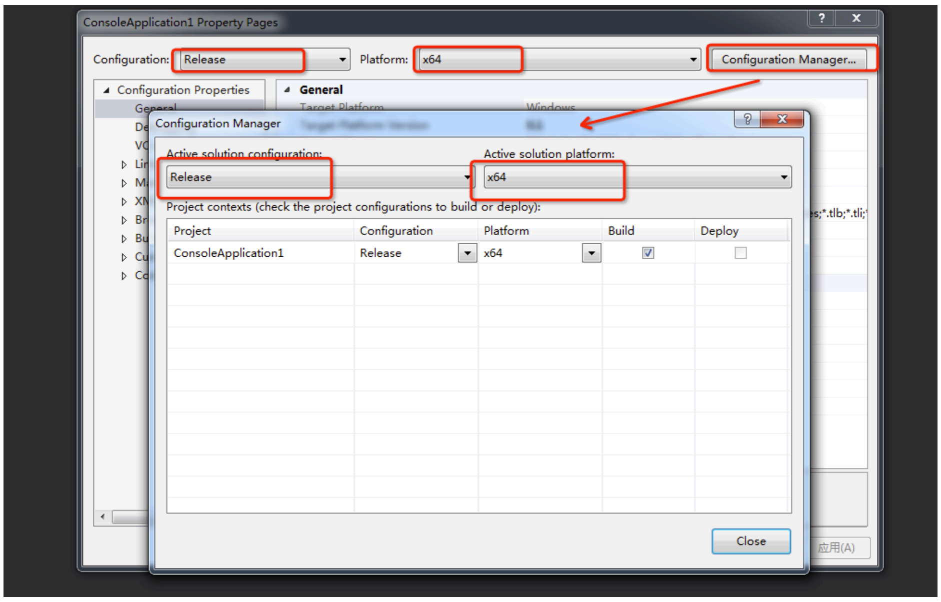
Add common header files for:

☐ ATL

☐ MFC

Right-click the project, choose **Properties** > **Configuration Manager**, and set Active Solution Configuration to **release** and Active Solution platform to **x64**.


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'ConsoleApplication1' (1 project)
Right click the project
Build
Rebuild
Clean
View
Analyze
Project Only
Retarget SDK Version
Scope to This
New Solution Explorer View
Build Dependencies
Add
Class Wizard... Ctrl+Shift+X
Manage NuGet Packages...
Set as StartUp Project
Debug
Cut Ctrl+X
Paste Ctrl+V
Remove Del
Rename
Unload Project
Rescan Solution
Open Folder in File Explorer
Properties Alt+Enter
Select Properties
Properties
ConsoleApplication1 Project Properties
(Name) ConsoleApplication1
(Name)
Specifies the project name.



ConsoleApplication1 Property Pages

Configuration: Release   Platform: x64   Configuration Manager...

Configuration Manager

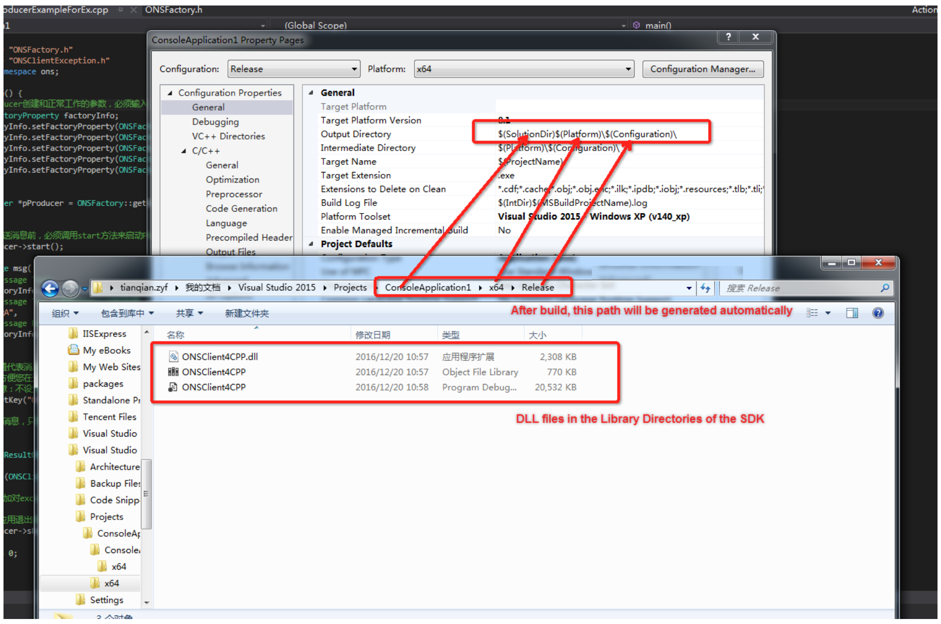
Active solution configuration: Release

Active solution platform: x64

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
ConsoleApplication1	Release	x64	✓	

Right-click the project, and choose **Properties** > **Configure Properties** > **General** > **Output Directory**: **/A**. Based on the setting of **Active Solution Platform**, copy all files in the 64-bit lib directory to the output directory **/A**.



ConsoleApplication1 Property Pages

Configuration: Release   Platform: x64   Configuration Manager...

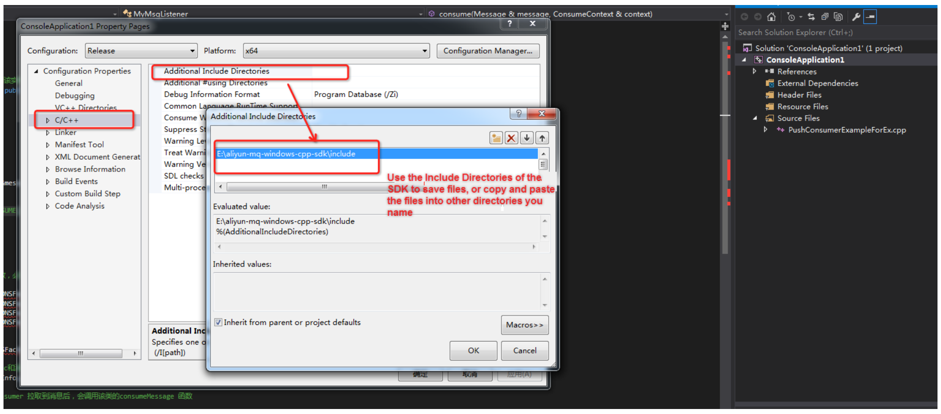
General	
Target Platform	
Target Platform Version	8.1
Output Directory	$(SolutionDir)$(Platform)\$(Configuration)\
Intermediate Directory	$(Platform)\$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 2015 - Windows XP (v140_xp)**
Enable Managed Incremental Build	No
Project Defaults	

tianqian.zyf ▸ 我的文档 ▸ Visual Studio 2015 ▸ Projects ▸ ConsoleApplication1 ▸ x64 ▸ Release

名称	修改日期	类型	大小
ONSClient4CPP.dll	2016/12/20 10:57	应用程序扩展	2,308 KB
ONSClient4CPP	2016/12/20 10:57	Object File Library	770 KB
ONSClient4CPP	2016/12/20 10:58	Program Debug...	20,532 KB

After build, this path will be generated automatically
DLL files in the Library Directories of the SDK

Right-click the property, and choose **Properties** > **Configure Properties** > **C/C++ - General** > **Additional Include Directories**: **/B**. Copy the header files in the include directory to the include directory **/B**.



ConsoleApplication1 Property Pages

Configuration: Release Platform: x64

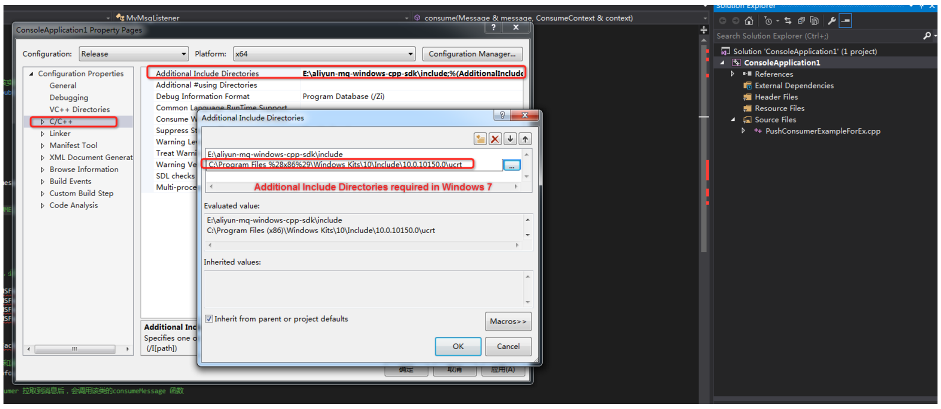
Configuration Properties > C/C++ > Additional Include Directories

Additional Include Directories:
E:\aliyun-mq-windows-cpp-sdk\include

Use the Include Directories of the SDK to save files, or copy and paste the files into other directories you name

Evaluated value:
E:\aliyun-mq-windows-cpp-sdk\include
%(AdditionalIncludeDirectories)

[✓] Inherit from parent or project defaults



Configuration: Release Platform: x64

Additional Include Directories: E:\aliyun-mq-windows-cpp-sdk\include;%(AdditionalInclude

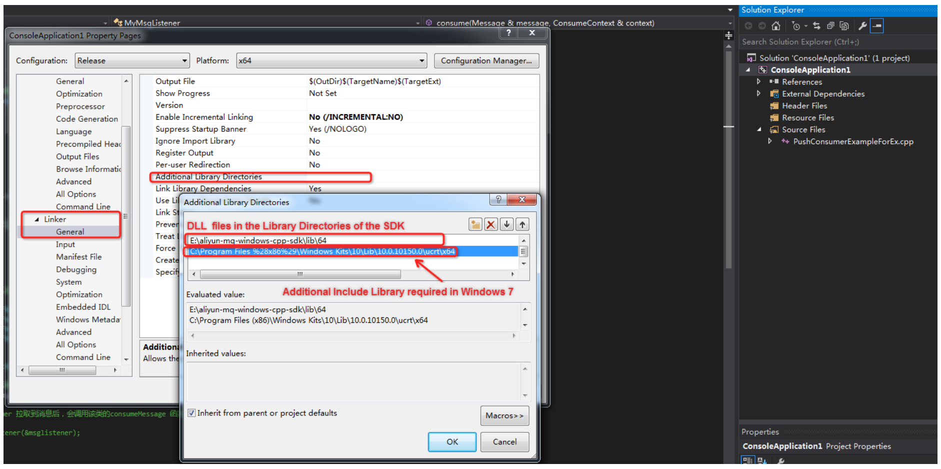
Additional Include Directories:
E:\aliyun-mq-windows-cpp-sdk\include
C:\Program Files %28x86%29\Windows Kits\10\Include\10.0.10150.0\ucrt

Additional Include Directories required in Windows 7

Evaluated value:
E:\aliyun-mq-windows-cpp-sdk\include
C:\Program Files (x86)\Windows Kits\10\Include\10.0.10150.0\ucrt

[✓] Inherit from parent or project defaults

Right-click the project, and choose **Properties** > **Configure Properties** > **Linker** > **General** > **Additional Library Directories**: **/A**.



Configuration: Release Platform: x64

Linker > General

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	No (/INCREMENTAL:NO)
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	
Link Library Dependencies	Yes

Additional Library Directories:

DLL files in the Library Directories of the SDK

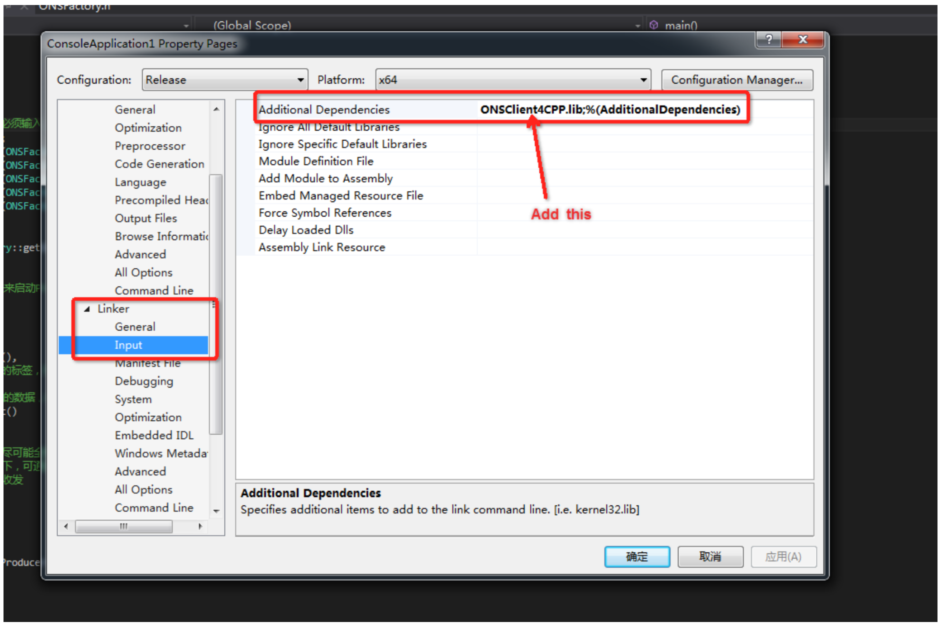
E:\aliyun-mq-windows-cpp-sdk\lib\64
C:\Program Files %28x86%29\Windows Kits\10\Lib\10.0.10150.0\ucrt\x64

Additional Include Library required in Windows 7

Evaluated value:
E:\aliyun-mq-windows-cpp-sdk\lib\64
C:\Program Files (x86)\Windows Kits\10\Lib\10.0.10150.0\ucrt\x64

[✓] Inherit from parent or project defaults

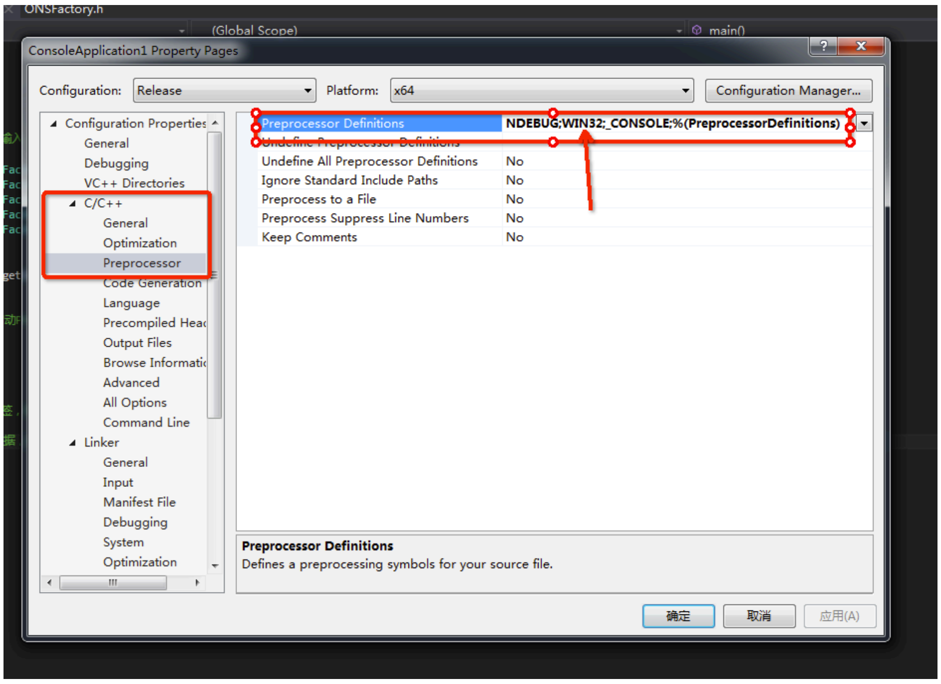
Right-click the project, and choose **Properties** > **Configure Properties** > **Linker** > **Input** > **Additional Dependencies**: **ONSClient4CPP.lib**.



Additional Dependencies: ONSClient4CPP.lib;%(AdditionalDependencies)

Additional Dependencies
Specifies additional items to add to the link command line. [i.e. kernel32.lib]

Right-click the project, and choose **Properties** > **Configure Properties** > **C/C++ - General** > **Preprocessor Definitions**: Add the**WIN32** macro.



Preprocessor Definitions	NDEBUG;WIN32;_CONSOLE;%(PreprocessorDefinitions)
Undefine Preprocessor Definitions	
Undefine All Preprocessor Definitions	No
Ignore Standard Include Paths	No
Preprocess to a File	No
Preprocess Suppress Line Numbers	No
Keep Comments	No

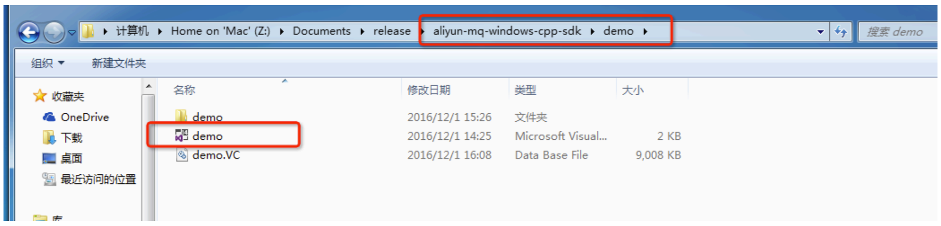
Preprocessor Definitions
Defines a preprocessing symbols for your source file.

Use C++ SDK in non-Visual Studio 2015 environment

Follow the preceding steps to configure your project based on the Visual Studio 2015 environment.

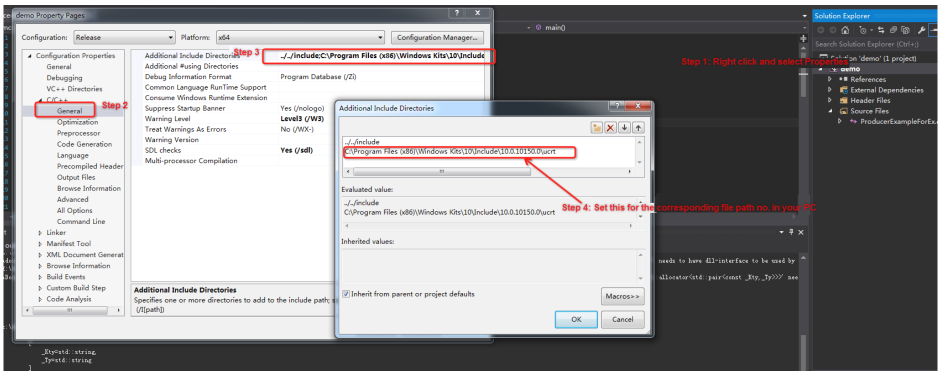
Install **vc_redist.x64**. This is the runtime environment for **Visual C++ 2015**.

Note: To avoid implementing complex settings, use the SDK demo that has already been set up. Download and decompress the SDK package, go to the demo directory, and run the project with Visual Studio 2015.



计算机 ▸ Home on 'Mac' (Z:) ▸ Documents ▸ release ▸ aliyun-mq-windows-cpp-sdk ▸ demo ▸

名称	修改日期	类型	大小
demo	2016/12/1 15:26	文件夹	
demo	2016/12/1 14:25	Microsoft Visual...	2 KB
demo.VC	2016/12/1 16:08	Data Base File	9,008 KB

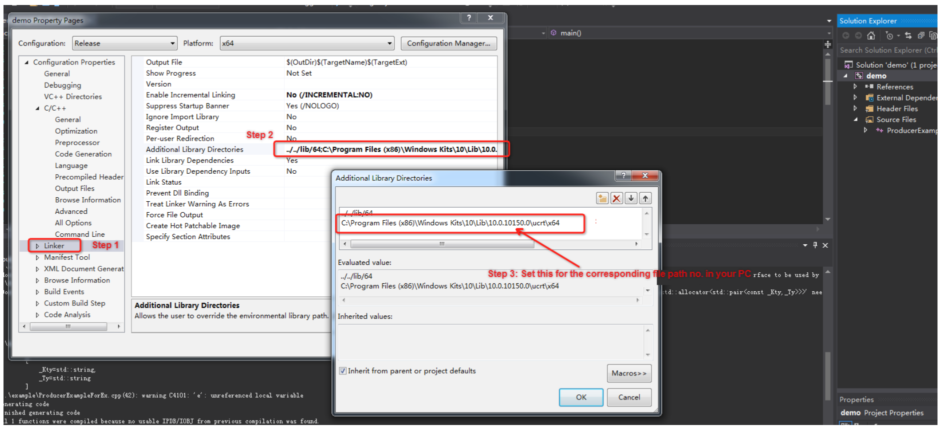

Step 1: Right click and select Properties
Step 2
Step 3
Step 4: Set this for the corresponding file path no. in your PC

demo Property Pages — Configuration: Release — Platform: x64

Configuration Properties ▸ C/C++ ▸ General

Additional Include Directories	../../include;C:\Program Files (x86)\Windows Kits\10\Include
Additional #using Directories	
Debug Information Format	Program Database (/Zi)
Common Language RunTime Support	
Consume Windows Runtime Extension	
Suppress Startup Banner	Yes (/nologo)
Warning Level	Level3 (/W3)
Treat Warnings As Errors	No (/WX-)
Warning Version	
SDL checks	Yes (/sdl)
Multi-processor Compilation	

Additional Include Directories:
```
../../include
C:\Program Files (x86)\Windows Kits\10\Include\10.0.10150.0\ucrt
```


Step 1
Step 2
Step 3: Set this for the corresponding file path no. in your PC

demo Property Pages — Configuration: Release — Platform: x64

Configuration Properties ▸ Linker

Output File	$(OutDir)$(TargetName)$(TargetExt)
Show Progress	Not Set
Version	
Enable Incremental Linking	No (/INCREMENTAL:NO)
Suppress Startup Banner	Yes (/NOLOGO)
Ignore Import Library	No
Register Output	No
Per-user Redirection	No
Additional Library Directories	../../lib/64;C:\Program Files (x86)\Windows Kits\10\Lib\10.0.
Link Library Dependencies	Yes
Use Library Dependency Inputs	No

Additional Library Directories:
```
../../lib/64
C:\Program Files (x86)\Windows Kits\10\Lib\10.0.10150.0\ucrt\x64
```

Now the compilation environment has been set up. Click **Build** to compile the executable application, and copy the DLL to the directory of the executable application or to the system directory to run the DLL.

Send and receive normal messages

Send Normal Messages

Refer to the example below to send messages.

```
#include "ONSFactory.h"
#include "ONSClientException.h"

using namespace ons;

int main()
{

//Create producer and configure parameters required for sending messages;
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );// Message content
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//Message content
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//AccessKey, Alibaba Cloud ID verification,
which is created on Alibaba Cloud Management Console
factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "XXX" );//SecretKey, Alibaba Cloud ID verification,
which is created on Alibaba Cloud Management Console

//create producer;
Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);

// Before sending messages, the start method must be called once to start the producer;
pProducer->start();

Message msg(
//Message Topic
factoryInfo.getPublishTopics(),
//Message tag, which is similar to tag in Gmail, and is used to classify messages. Consumers can then set filtering
conditions for messages to be filtered in MQ broker.
"TagA",
//Message Body, which cannot be null. Serialization and deserialization methods need to be negotiated and remain
consistent between the producer and the consumer.
factoryInfo.getMessageContent()
);

// The setting represents the key business property of the message, so please keep it globally unique.
// You can query a message and resend it through the MQ console when you cannot receive the message properly.
// Note: Normal sessage sending and receiving will not be affected if message key is not configured.
msg.setKey("ORDERID_100");

// If no exceptions are thrown, then the message is sent successfully.
try
{
SendResultONS sendResult = pProducer->send(msg);
}
catch(ONSClientException & e)
{
//Customize the details for processing the exception
}
// The object Producer must be destroyed before exiting the application. Otherwise there will be memory leakage.
pProducer->shutdown();

return 0;
}
```

Subscribe to Normal Messages

For instructions and sample codes of subscribing to standard messages, see Subscribe to messages.

Send and receive ordered messages

Send ordered messages

Sample code:

```
#include "ONSFactory.h"
#include "ONSClientException.h"
#include <iostream>
using namespace ons;

int main()
{
//parameter required for producer creation and message sending and receiving
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"xxxxxxxxxxxx" );// The msg topic created on MQ console
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "input msg content");//Message content
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxxxxxxxx");//MQ AccessKey
factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxxxxxxxxxxxxxxxxxx" );// MQ SecretKey
// Set TCP endpoint (This example is for the public cloud environment)
factoryInfo.setFactoryProperty(ONSFactoryProperty::ONSAddr, "http://onsaddr-internet.aliyun.com/MQ/nsaddr4client-internet")

//Create a producer
OrderProducer *pProducer = ONSFactory::getInstance()->createOrderProducer(factoryInfo);


//Before sending a message, the start method must be called once to start the producer.
pProducer->start();

Message msg(
//Message Topic
factoryInfo.getPublishTopics(),
//Message tag, which is similar to tag in Gmail, and is used to classify messages. Consumers can then set filtering conditions for messages to be filtered in MQ broker.
"TagA",
//Message body. Any binary data is allowed and MQ does not perform any processing on it. Serialization and deserialization methods need to be negotiated and remain consistent between the producer and the consumer.
factoryInfo.getMessageContent()
);
// The setting represents the key business property of the message, so please keep it globally unique.
// You can query a message and resend it through the MQ console when you cannot receive the message properly.
// Note: Normal sessage sending and receiving will not be affected if message key is not configured.
msg.setKey("ORDERID_100");
// For partitionally ordered messages, sharding keys are used to distinguish different partitions.
// For globally ordered messages, it can be set as any string that is not null.
std::string shardingKey = "abc"; //Messages with the same shardingKey will be sent in order.
try
{
//Send message. If no exceptions are thrown, then the message is sent successfully.
SendResultONS sendResult = pProducer->send(msg, key);
std::cout << "send success" << std::endl;
}
catch(ONSClientException & e)
{
//Handle exceptions
}
// You must destroy the object Producer before exiting the application. Otherwise there will be memory leakage.
pProducer->shutdown();

return 0;
}
```

Receive ordered messages

Sample code:

```


#include "ONSFactory.h"
using namespace std;
using namespace ons;

//Create a message consumer instance
//When pushConsumer has pulled the message, it will actively invoke the consumeMessage function of the
instance.
class ONSCLIENT_API MyMsgListener : public MessageOrderListener
{
public:
MyMsgListener()
{
}
virtual ~MyMsgListener()
{
}

virtual OrderAction consume(Message &message, ConsumeOrderContext &context)
{
//Consume messages, based on business requirements
return Success; //CONSUME_SUCCESS;
}
};


int main(int argc, char* argv[])
{
```

```
//OrderConsumer property required for creation and normal operation
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "");//The consumerId created on the console
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"" );// The msg topic created on the console
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "");// MQ AccessKey
factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "");//MQ SecretKey
// Set TCP endpoint (This example is for the public cloud environment)
factoryInfo.setFactoryProperty(ONSFactoryProperty::ONSAddr, "http://onsaddr-
internet.aliyun.com/MQ/nsaddr4client-internet");


// Create orderConsumer
OrderConsumer* orderConsumer = ONSFactory::getInstance()->createOrderConsumer(factoryInfo);
MyMsgListener msglistener;
//Specify the topics and tags that the orderConsumer subscribes to
orderConsumer->subscribe(factoryInfo.getPublishTopics(), "*",&msglistener );

// Register the message listener processing instance. After orderConsumer pulls the message, it calls the
consumeMessage method of this class.

//Start orderConsumer
orderConsumer->start();

for(volatile int i = 0; i < 1000000000; ++i) {
//wait
}

//Destroy orderConsumer. You must be destroy the object Consumer before exiting the application, otherwise there
will be memory leaks.
orderConsumer->shutdown();

return 0;
}
```

Send and receive scheduled messages

The currently supported regions include Internet, East China 1, East China 2, North China 2, and South China 1.

Scheduled messages can be consumed by consumers after a specified period, which are used in scenarios where there are time window requirements for message production and consumption, or when messages are used to trigger scheduled tasks, similar to delayed queues.

Send Scheduled Messages

The following are sample codes for sending scheduled messages:

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;
int main()
{

//Create producer and configure parameters required for sending messages;
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//The topic you created on the MQ
console
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "xxx");//msg content
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxx");//Alibaba Cloud ID verification, which is
created on Alibaba Cloud Management Console
factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxx" );//Alibaba Cloud ID verification, which is
created on Alibaba Cloud Management Console

//create producer
Producer *pProducer = ONSFactory::getInstance()->createProducer(factoryInfo);

// Before sending messages, the start method must be called once to start the producer;
pProducer->start();

Message msg(
// Message topic
factoryInfo.getPublishTopics(),
//Message tag, which is similar to tag in Gmail, and is used to classify messages. Consumers can then set filtering
conditions for messages to be filtered in MQ broker.
"TagA",
//Message Body, which cannot be null. Serialization and deserialization methods need to be negotiated and remain
consistent between the producer and the consumer.
factoryInfo.getMessageContent()
);

// The setting represents the key service property of the message, so please set it as globally unique as possible.
// You can query a message and resend it through the MQ console when you cannot receive the message properly.
// Note: Normal sessage sending and receiving will not be affected if message key is not configured.
msg.setKey("ORDERID_100");

// Deliver time (ms) specifies the time point after which the message can be consumed. The example means that
the message will be consumed after 3 seconds.
long deliverTime = obtain current system time (ms) + 3000;
msg.setStartDeliverTime(deliverTime);

// If no exceptions are thrown, then the message is sent successfully.
try
{
SendResultONS sendResult = pProducer->send(msg);
}
catch(ONSClientException e)
{
//Customize the details for processing exceptions
}

// The object Producer must be destroyed before exiting the application. Otherwise there will be memory leakage.
pProducer->shutdown();
return 0;
}
```

Subscribe to Scheduled Messages

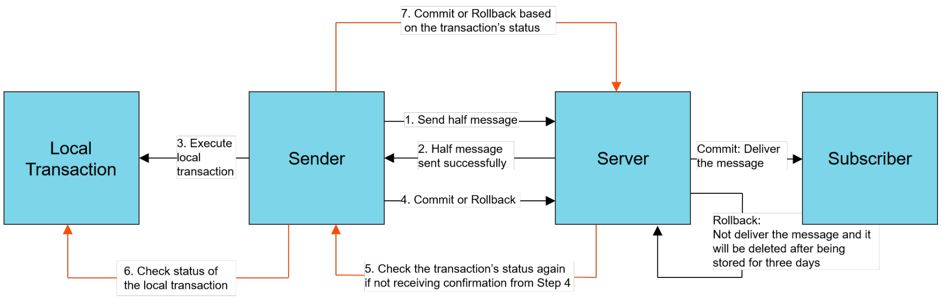
For instructions and example codes of subscribing to scheduled messages, see Subscribe Message.

Send and receive transactional messages

The currently supported regions include Internet, China (Hangzhou), China (Beijing), China (Shanghai), and China (Shenzhen).

Interaction process

The interaction process of transactional messages is shown in the following figure.


7. Commit or Rollback based on the transaction's status
Local Transaction
3. Execute local transaction
Sender
1. Send half message
2. Half message sent successfully
4. Commit or Rollback
Server
Commit: Deliver the message
Subscriber
Rollback: Not deliver the message and it will be deleted after being stored for three days
6. Check status of the local transaction
5. Check the transaction's status again if not receiving confirmation from Step 4

Send transactional messages

Follow these steps to send a transactional message:

1. Send a half message and execute a local transaction. The sample code is as follows:

```
#include "ONSFactory.h"
#include "ONSClientException.h"
using namespace ons;

class MyLocalTransactionExecuter : LocalTransactionExecuter
{
MyLocalTransactionExecuter()
{
}

~MyLocalTransactionExecuter()
```

```
{
}
virtual TransactionStatus execute(Message &value)
{
// The message ID. Two messages may have the same message body but different message IDs. The current
message ID cannot be obtained in the console.
string msgId = value.getMsgID();
// Compute the message body by using CRC32, MD5, or other algorithms.
// The message ID and CRC32 ID are used to prevent duplication of messages.
// You do not need to specify the message ID or CRC32 ID if the business itself achieves idempotence. Otherwise,
set the message ID or CRC32 ID to ensure idempotence.
// To avoid duplication of messages, compute the message body by using the CRC32 or MD5 algorithm.
TransactionStatus transactionStatus = Unknow;
try {
boolean isCommit = Execution result of the local transaction
if (isCommit) {
// If the local transaction succeeded, the message is submitted.
transactionStatus = CommitTransaction;
} else {
// If the local transaction failed, the message is rolled back.
transactionStatus = RollbackTransaction;
}
} catch (...) {
//exception handle
}
return transactionStatus;
}
}

int main(int argc, char* argv[])
{
//The information that is required to create a producer and to send messages.
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ProducerId,"XXX");//The group ID you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //Set the TCP endpoint: Go to the
**Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//The topic you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty::MsgContent, "XXX");//msg content
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "xxx");//The AccessKeyId that was created in the
Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "xxxxxxxxxxxxxxxxxxxx" );//The AccessKeySecret that
was created in the Alibaba Cloud console for identity authentication.

//Create a producer. MQ does not release pChecker, which must be released by the service provider.
MyLocalTransactionChecker *pChecker = new MyLocalTransactionChecker();
g_producer = ONSFactory::getInstance()->createTransactionProducer(factoryInfo,pChecker);

//Before sending a message, call the start method once to start the producer.
pProducer->start();

Message msg(
//The message topic.
factoryInfo.getPublishTopics(),
// The message tag, which is similar to a Gmail tag. It is used to sort messages, enabling the consumer to filter
messages on the MQ broker based on the specified criteria.
"TagA",
//The body of the message. The message body cannot be empty. MQ makes no interventions. The compatible serialization and deserialization methods must be negotiated by the producer and the consumer.
factoryInfo.getMessageContent()
);

// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the console if you fail to receive the message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_100");

// The message is sent if no exception is thrown.
try
{
//MQ does not release pExecuter, which must be released by the service provider.
MyLocalTransactionExecuter pExecuter = new MyLocalTransactionExecuter();
SendResultONS sendResult = pProducer->send(msg,pExecuter);
}
catch(ONSClientException & e)
{
//Customize the exception handling details.
}
// Destroy the producer before exiting the application. Otherwise, memory leakage may occur.
pProducer->shutdown();

return 0;

}
```

1. Submit the status of the transactional message.

After the execution of a local transaction (successful or failed), the broker must be notified of the transaction status of the current message. Two notification modes are supported:

- Submit the status after executing the local transaction.

- Wait until the broker requests to check the transaction status of the message.

A transaction may be in one of the following states:

- TransactionStatus.CommitTransaction: The transaction is submitted, and the consumer can consume the message.

- TransactionStatus.RollbackTransaction: The transaction is rolled back, and the message is discarded and cannot be consumed.

- TransactionStatus.Unknow: The transaction is in an unknown status, and the broker is expected to query the status of the local transaction that corresponds to the message from the message sender.

```
 class MyLocalTransactionChecker : LocalTransactionChecker
{
MyLocalTransactionChecker()
{
}

~MyLocalTransactionChecker()
{
}

virtual TransactionStatus check(Message &value)
{
// The ID of the message. Two messages may have the same body but different IDs. Currently, you cannot
query message IDs in the console.
string msgId = value.getMsgID();
// Compute the message body by using CRC32, MD5, or other algorithms.
// The message ID and CRC32 ID are used to prevent duplication of messages.
// You do not need to specify the message ID or CRC32 ID if the business itself achieves idempotence.
Otherwise, set the message ID or CRC32 ID to ensure idempotence.
// To avoid duplication of messages, compute the message body by using the CRC32 or MD5 algorithm.
TransactionStatus transactionStatus = Unknow;
try {
boolean isCommit = Execution result of the local transaction
if (isCommit) {
// If the local transaction succeeded, the message is submitted.
transactionStatus = CommitTransaction;
} else {
// If the local transaction failed, the message is rolled back.
transactionStatus = RollbackTransaction;
}
} catch(...) {
//exception error
}
return transactionStatus;
}
}
```

Transaction check mechanism

Why must the transaction status check mechanism be implemented when transactional messages are sent?

When a half message is sent in step 1, but the returned status of the local transaction is TransactionStatus.Unknow, or no status is submitted because the application exits, the status of the half message is unknown to the MQ broker. Therefore, the MQ broker requires the message sender to check the status of the half message and to periodically report the final status.

What does the business logic do when the check method is called back?

The check method for transactional messages needs to contain the logic of transaction consistency check. After a transactional message is sent, MQ needs to use the LocalTransactionChecker operation to respond to the request of the broker for the local transaction status. Therefore, the check method for the transactional message needs to complete the following tasks:

(1) Check the status of the local transaction corresponding to the half message (committed or rollback).

(2) Submit the status of the local transaction to the broker.

What is the impact of different local transaction statuses on the half message?

TransactionStatus.CommitTransaction: The transaction is submitted, and the consumer can consume the message.

TransactionStatus.RollbackTransaction: The transaction is rolled back, and the message is discarded and cannot be consumed.

TransactionStatus.Unknow: The transaction is in an unknown status, and the broker is expected to query the status of the local transaction that corresponds to the message from the message sender.

For more information about the code, see the implementation of MyLocalTransactionChecker.

Subscribe to transactional messages

For instructions and sample codes for subscribing to normal messages, see Subscribe to messages.

Subscribe to messages

This topic describes how to subscribe to messages by using the C/C++ SDK of MQ.

Note:

- Maintain consistent subscription for all consumer instances with the same group ID. For more information, see Subscription consistency.

Subscription modes

MQ supports the following two message subscription modes:

- **Clustering subscription:**

Clustering consumption is realized in this mode. All consumers that are identified by the same group ID consume messages in an even manner. For example, a topic contains nine messages and a group contains three consumer instances. In this case, each instance consumes three messages.

```
```
// Set the subscription mode to clustering. Clustering subscription is used by default when this parameter is not configured.
factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::CLUSTERING);
```
```

- **Broadcasting subscription:**

Broadcasting subscription is implemented in this mode. Each of the consumers that is identified by the same group ID consumes all messages once. For example, a topic contains nine messages and a group contains three consumer instances. In this case, each instance consumes nine messages.

```
```
// Set the subscription mode to broadcasting.
factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::BROADCASTING);
```
```

Sample code

```
#include "ONSFactory.h"
using namespace ons;

// MyMsgListener: Create a message consumer instance.
//When pushConsumer pulls the message, it actively calls the consumer function of the instance.
class MyMsgListener : public MessageListener
{

public:

MyMsgListener()
{
}

virtual ~MyMsgListener()
{
}

virtual Action consume(Message &message, ConsumeContext &context)
{
//Customize the details of the message processing policy.
return CommitMessage; //CONSUME_SUCCESS;
}
};


int main(int argc, char* argv[])
{

//Required parameter for creating and executing pushConsumer.
ONSFactoryProperty factoryInfo;
factoryInfo.setFactoryProperty(ONSFactoryProperty::ConsumerId, "XXX");//The group ID you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty::NAMESRV_ADDR, "XXX"); //Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
factoryInfo.setFactoryProperty(ONSFactoryProperty::PublishTopics,"XXX" );//The message topic you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty::AccessKey, "XXX");//The AccessKeyId you created in the Alibaba Cloud console for identify authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty::SecretKey, "XXX");//The AccessKeySecret you created in the Alibaba Cloud console for identify authentication.
// Set the subscription mode to clustering, which is the default subscription mode.
// factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::CLUSTERING);
//Set the subscription mode to broadcasting.
// factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty::BROADCASTING);

//create pushConsumer
PushConsumer* pushConsumer = ONSFactory::getInstance()->createPushConsumer(factoryInfo);

//Specify the topic and tag of the message to which pushConsumer subscribes, and register the message callback function.
MyMsgListener msglistener;
pushConsumer->subscribe(factoryInfo.getPublishTopics(), "*",&msglistener );

//start pushConsumer
pushConsumer->start();

//Note: Shutdown can be called only after the consumer no longer receives messages. After shutdown is called, the consumer exits and cannot receive any message any more.

//Destroy pushConsumer. You must destroy the consumer before exiting the application. Otherwise, memory leakage may occur.
pushConsumer->shutdown();
return 0;

}
```

.NET SDK

Release Notes

This topic provides the download links, versions, and updates of all .NET SDKs so that you can choose a suitable one for use.

ons-.net v1.1.3

Release date	Version	Download (Windows version)	Environment preparation guide
2019-02-01	1.1.3	aliyun-mq-windows-net-sdk.zip	. Prepare the .NET SDK environment

New features

Enabled instance user access to the service in either of the following modes (that for non-instance users unchanged):

Configure NAMESRV_ADDR with InstanceId.

Configure InstanceId and NAMESRV_ADDR without InstanceId.

Replaced ProducerId and ConsumerId with GroupId.

ons-.net v1.1.2

Release date	Version	Download (Windows version)	Environment preparation guide
2018-10-24	1.1.2	aliyun-mq-windows-net-sdk.zip	. Prepare the .NET SDK environment

Function optimization

- Provided the recommended Chinese encoding mode in the demo.

Bugs fixed

- Fixed the problem of consumption of ordered messages.

More historical versions

ons-.net v1.1.1

Release date	Version	Earlier version download (Windows)	New version download (Windows)	Environment preparation guide
2018-01-09	1.1.1	None (not maintained anymore)	aliyun-mq-windows-net-sdk.zip	. Prepare the .NET SDK environment

New features

- Added an operation for sending byte messages.

Bugs fixed

- Fixed the problem that the IP address obtained from the message trace is incorrect.

ons-.net v1.1.0

Release date	Version	Earlier version download (Windows)	New version download (Windows)	Environment preparation guide
2017-07-31	1.1.0	aliyun-mq-windows-net-sdk.zip	aliyun-mq-windows-net-sdk.zip	. Prepare the .NET SDK environment

Bugs fixed

- Fixed coredump caused by consumer shutdown.
- Fixed the problem that the underlying URL class does not support HTTP access on Windows.
- Fixed the timestamp error of message trace.
- Fixed the problem that an incorrect IP address is displayed in message trace.
- Fixed the problem of memory leakage on Windows.

Prepare the .NET SDK environment

Complete the following preparations before accessing MQ through .NET SDK.

Note:

The topic and group ID in the code must have been created in the MQ console first. The message tag can be specified by the application users. For more information about the creation process, see **Step 2: Create resources** in Quick start for primary accounts.

Applications that use MQ must be deployed on Alibaba Cloud ECS instances.

Download SDK

Windows .NET SDK

The .NET SDK we provide is based on the managed wrapper of MQ C++. In this way, the .NET SDK is independent of Windows .NET SDK public libraries. C++ multi-thread concurrent processing is used to ensure the efficiency and stability of the .NET SDK.

When Visual Studio is used to develop .NET applications and class libraries, the default target platform is “Any CPU”, that is, X86 or X64 is automatically selected according to the CPU type when the application runs. When running, CLR transmits its JIT as the machine code X86 or X64. The C or C++ compiled DLL is the machine code. Accordingly, the policies of the platform are determined when the application is compiled. When the compilation options are set, the C/C++ project is compiled as an X64 64-bit DLL. Therefore, the 64-bit DLL in release mode compiled using Visual Studio 2015 is provided. Other Visual Studio versions also can be used.

Download earlier versions of Windows .NET SDK

Note:

- The SDK based on the managed wrapper has a lot of problems, and cannot normally operate on ASP.NET. Therefore, a new version SDK was released on December 29, 2016.
- The new SDK calls underlying DLLs based on C# PInvoke and uses the open-source software SWIG to generate the PInvoke wrapped code. Compared with the managed SDK, the new SDK is more stable and easier to deploy and install.
- The managed SDK is no longer maintained. Only the latest stable version is provided.

We recommend that both new users and old users download the new SDK.

For the URL to download the latest .NET SDK, see Release Notes.

Download and decompress the .rar package of the .NET SDK. The .rar package contains the following directories and files:

- example/
- lib/
- demo/
- interface/
- SDK_GUIDE.pdf
- changelog

The preceding directories and files serve the following purposes:

example: This folder contains examples for sending and consuming normal messages and ordered messages, and examples for sending messages in one-way mode.

lib: This folder contains the C++ DLL files and the runtime installation package for **Virtual C++ 2015**.

```
64/
NSClient4CPP.lib
ONSClient4CPP.dll
ONSClient4CPP.pdb
vc_redist.x64.exe
```

SDK_GUIDE.pdf: This file describes how to prepare the SDK environment and contains FAQ.

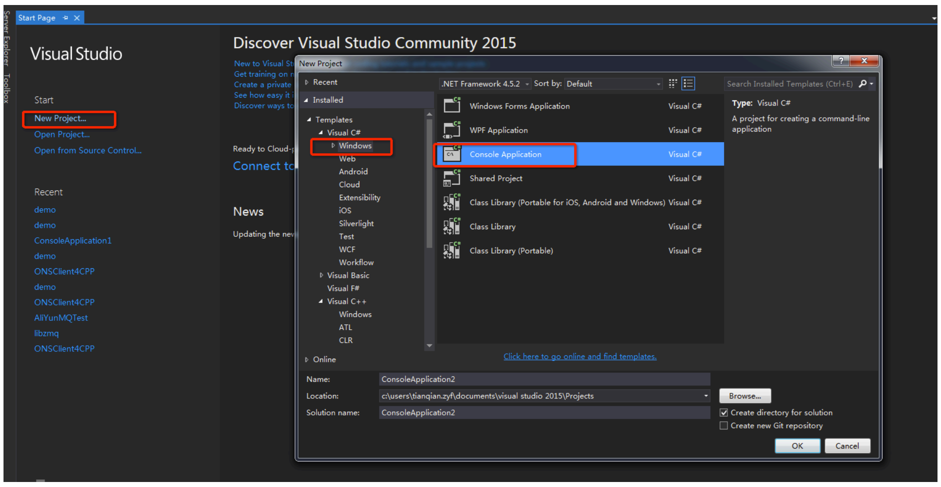
changelog: This file lists the problems that have been fixed and the new features of the new version.

interface: This folder contains codes that are called to wrap PInvoke, which needs to be used in the user project code.

.NET SDK configuration

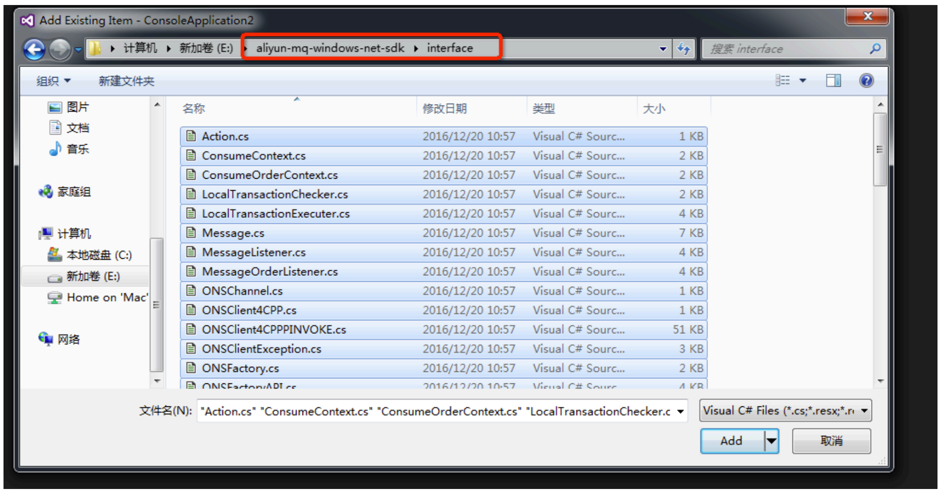
Configuration for using .NET SDK in Visual Studio 2015

Use **Visual Studio 2015** to create your project.



Name: ConsoleApplication2
Location: c:\users\tianqian.zyf\documents\visual studio 2015\Projects
Solution name: ConsoleApplication2
☑ Create directory for solution
☐ Create new Git repository

Right-click the project and choose **Add** > **Existing Item**to add all the files in the **interface** folder in the downloaded SDK package.


Add Existing Item - ConsoleApplication2

计算机 ▸ 新加卷 (E:) ▸ aliyun-mq-windows-net-sdk ▸ interface

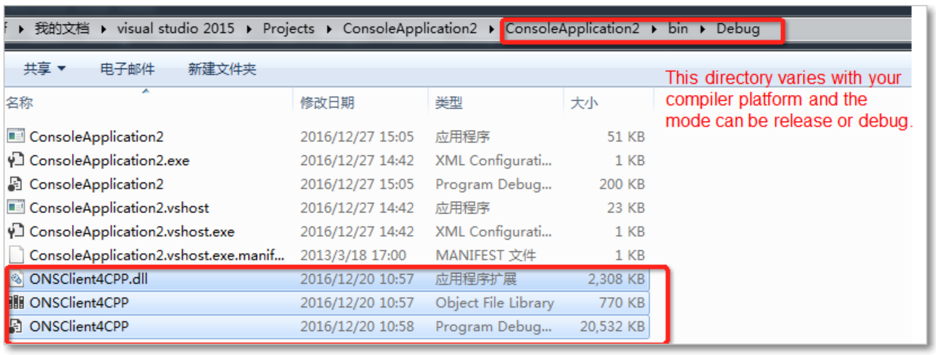
名称	修改日期	类型	大小
Action.cs	2016/12/20 10:57	Visual C# Sourc...	1 KB
ConsumeContext.cs	2016/12/20 10:57	Visual C# Sourc...	2 KB
ConsumeOrderContext.cs	2016/12/20 10:57	Visual C# Sourc...	2 KB
LocalTransactionChecker.cs	2016/12/20 10:57	Visual C# Sourc...	2 KB
LocalTransactionExecuter.cs	2016/12/20 10:57	Visual C# Sourc...	4 KB
Message.cs	2016/12/20 10:57	Visual C# Sourc...	7 KB
MessageListener.cs	2016/12/20 10:57	Visual C# Sourc...	4 KB
MessageOrderListener.cs	2016/12/20 10:57	Visual C# Sourc...	4 KB
ONSChannel.cs	2016/12/20 10:57	Visual C# Sourc...	1 KB
ONSClient4CPP.cs	2016/12/20 10:57	Visual C# Sourc...	1 KB
ONSClient4CPPPINVOKE.cs	2016/12/20 10:57	Visual C# Sourc...	51 KB
ONSClientException.cs	2016/12/20 10:57	Visual C# Sourc...	3 KB
ONSFactory.cs	2016/12/20 10:57	Visual C# Sourc...	2 KB

文件名(N): "Action.cs" "ConsumeContext.cs" "ConsumeOrderContext.cs" "LocalTransactionChecker.c

Visual C# Files (*.cs;*.resx;*.r

Right-click the project and choose **Properties** > **Configuration Manager**. Set **Active Solution Configuration** to **Release**, and **Active Solution Platform** to **x64**.

Write a test program and compile it. Move the DLL in the SDK to the same directory of the executable files or to the system directory. Then, the DLL can be executed.

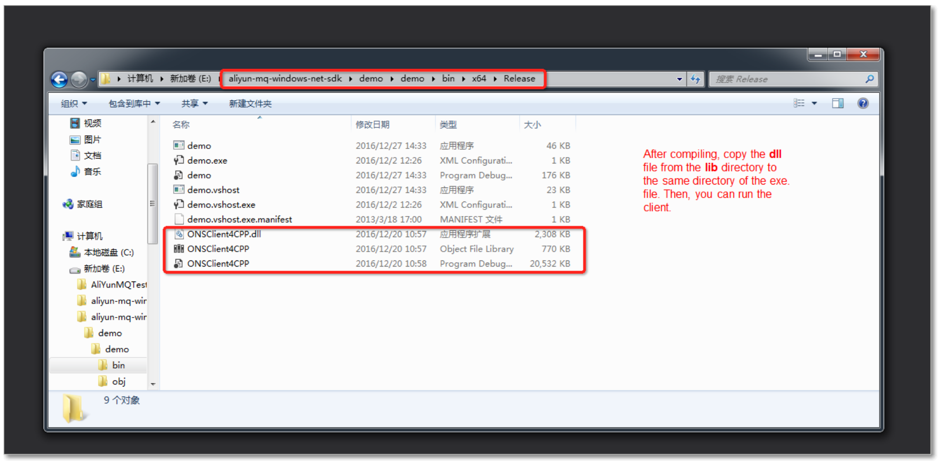

我的文档 ▸ visual studio 2015 ▸ Projects ▸ ConsoleApplication2 ▸ ConsoleApplication2 ▸ bin ▸ Debug

名称	修改日期	类型	大小
ConsoleApplication2	2016/12/27 15:05	应用程序	51 KB
ConsoleApplication2.exe	2016/12/27 14:42	XML Configurati...	1 KB
ConsoleApplication2	2016/12/27 15:05	Program Debug...	200 KB
ConsoleApplication2.vshost	2016/12/27 14:42	应用程序	23 KB
ConsoleApplication2.vshost.exe	2016/12/27 14:42	XML Configurati...	1 KB
ConsoleApplication2.vshost.exe.manif...	2013/3/18 17:00	MANIFEST 文件	1 KB
ONSClient4CPP.dll	2016/12/20 10:57	应用程序扩展	2,308 KB
ONSClient4CPP	2016/12/20 10:57	Object File Library	770 KB
ONSClient4CPP	2016/12/20 10:58	Program Debug...	20,532 KB

共享
电子邮件
新建文件夹
This directory varies with your compiler platform and the mode can be release or debug.

Note:

The SDK provides a demo that has been set up, and you can open the project and compile it. When running the SDK, copy the DLL files to the same directory of the executable files, as shown in the following figure:

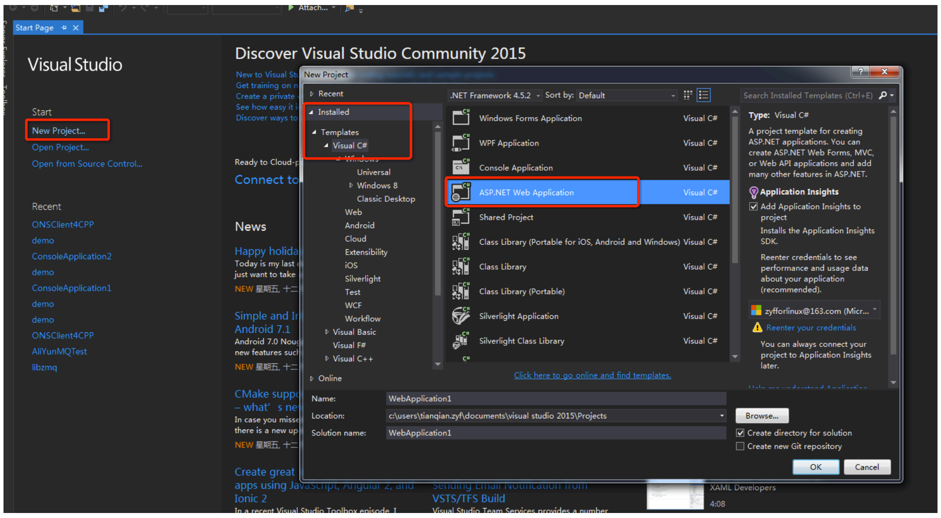


After compiling, copy the **dll** file from the **lib** directory to the same directory of the exe file. Then, you can run the client.

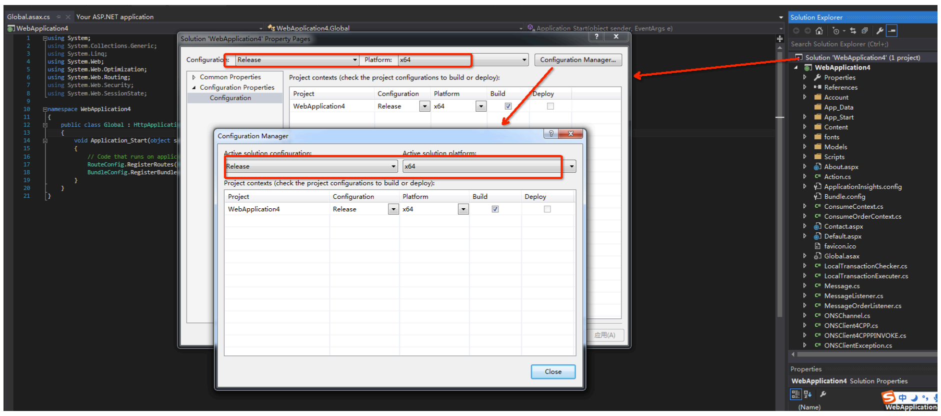
名称	修改日期	类型	大小
demo	2016/12/27 14:33	应用程序	46 KB
demo.exe	2016/12/2 12:26	XML Configurati...	1 KB
demo	2016/12/27 14:33	Program Debug...	176 KB
demo.vshost	2016/12/27 14:33	应用程序	23 KB
demo.vshost.exe	2016/12/2 12:26	XML Configurati...	1 KB
demo.vshost.exe.manifest	2013/3/18 17:00	MANIFEST 文件	1 KB
ONSClient4CPP.dll	2016/12/20 10:57	应用程序扩展	2,308 KB
ONSClient4CPP	2016/12/20 10:57	Object File Library	770 KB
ONSClient4CPP	2016/12/20 10:58	Program Debug...	20,532 KB

Configuration for using MQ SDK to create ASP.NET in Visual Studio 2015

Create an ASP.NET **Web Forms** project by using Visual Studio 2015.


New Project...
Installed
Templates
Visual C#
ASP.NET Web Application
Name: WebApplication1
Location: c:\users\tianqian.zyf\documents\visual studio 2015\Projects
Solution name: WebApplication1
OK
Cancel

Right-click the project and choose **Properties** > **Configuration Manager**. Set **Active Solution Configuration** to **Release**, and **Active Solution Platform** to **x64**.


Configuration Manager
Active solution configuration: Release
Active solution platform: x64
WebApplication4 Release x64
Close

Right-click the project and choose **Add** > **Existing Item** to add all the files in the interface folder in the downloaded SDK package.

See the preceding step 2 for configuring the normal .NET project.

1. In the file **Global.asax.cs**, add the codes for enabling and disabling the SDK.

Note:

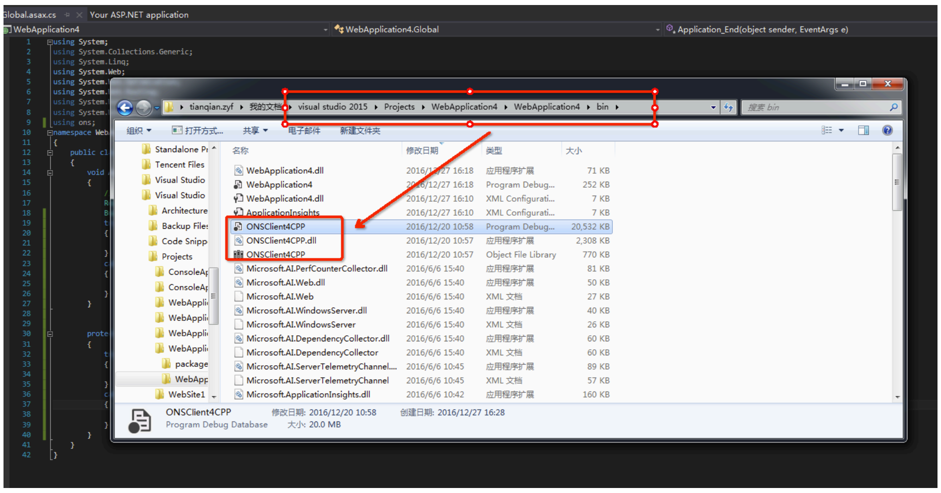
We recommend that you wrap the SDK code to a singleton class, to prevent the SDK code from being collected by the garbage collector due to scope issues. The folder example in the SDK provides Example.cs, which implements a simple singleton. To use Example.cs, you need to include Example.cs in your project.

~~~
```csharp
~~~

sing System;using System.Collections.Generic;using System.Linq;using System.Web;using System.Web.Optimization;using System.Web.Routing;using System.Web.Security;using System.Web.SessionState;using ons; //It is the namespace where the SDK resides.using test; // It is the namespace where a wrapped SDK class resides. For more information, see Example.cs in the folder example in the SDK.namespace WebApplication4{public class Global : HttpApplication{void Application_Start(object sender, EventArgs e){// Code that runs on application startupRouteConfig.RegisterRoutes(RouteTable.Routes);BundleConfig.RegisterBundles(BundleTable.Bundles);try{// The code for starting the SDK. The following is a code after the SDK is wrapped into a simple singleton.OnscSharp.CreateProducer();OnscSharp.StartProducer();}catch (Exception ex){//Handle exceptions.}}protected void Application_End(object sender, EventArgs e){try{// The code for disabling the SDK.OnscSharp.ShutdownProducer();}catch (Exception ex){// Handle exceptions.}}}}```

Write a test program and compile it.

Move the DLL in the SDK to the same directory of the executable files or to the system directory. Then, the DLL can be executed.

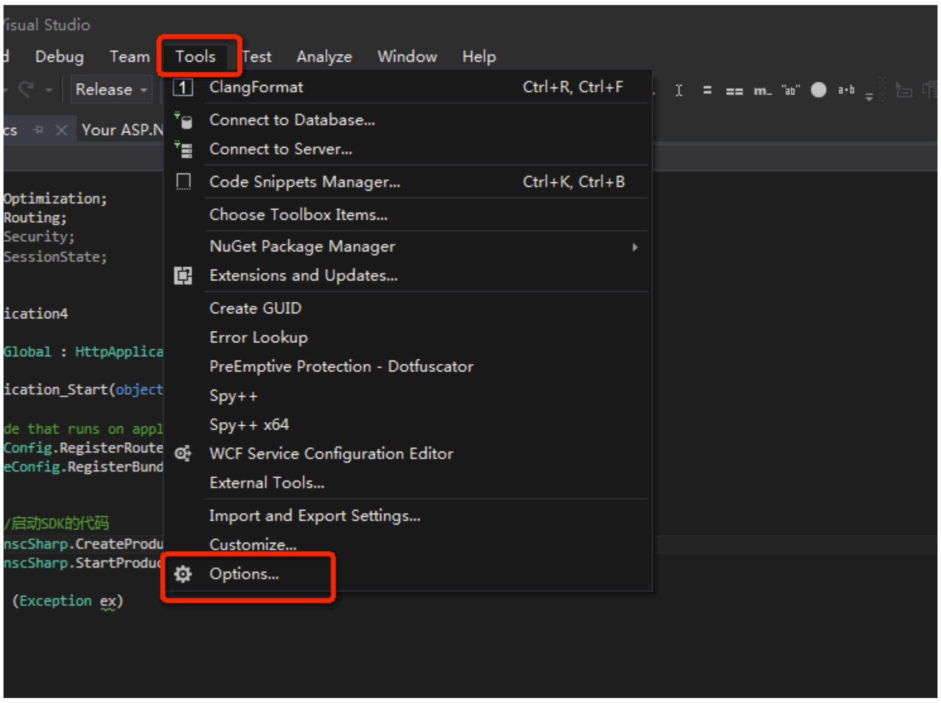


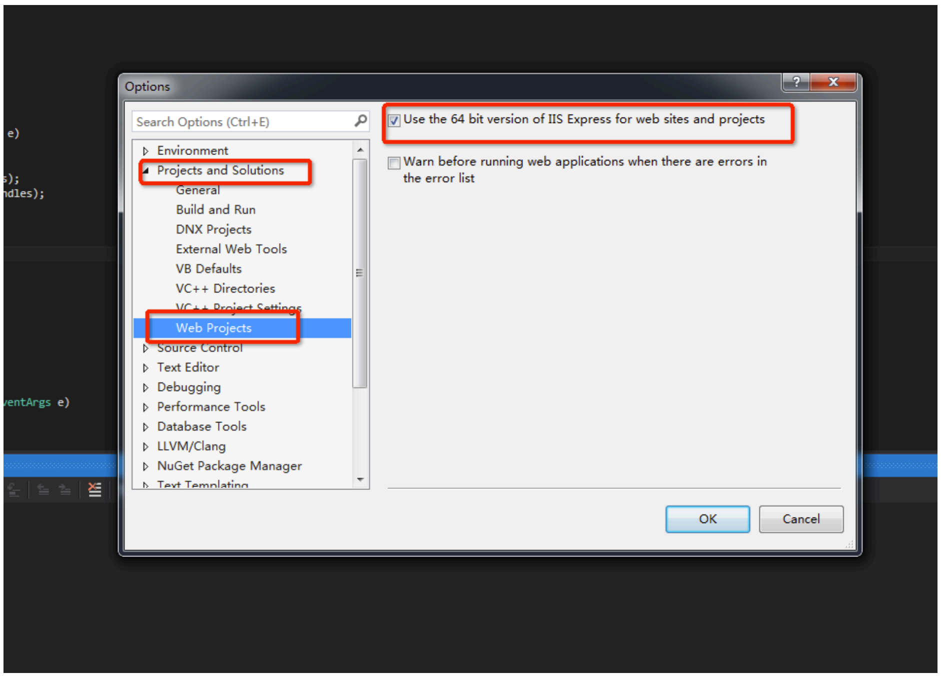
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.
using System.
using System.
using System.
using ons;
namespace Web
```

名称	修改日期	类型	大小
WebApplication4.dll	2016/12/27 16:18	应用程序扩展	71 KB
WebApplication4	2016/12/27 16:18	Program Debug...	252 KB
WebApplication4.dll	2016/12/27 16:10	XML Configurati...	7 KB
ApplicationInsights	2016/12/27 16:10	XML Configurati...	7 KB
ONSClient4CPP	2016/12/20 10:58	Program Debug...	20,532 KB
ONSClient4CPP.dll	2016/12/20 10:57	应用程序扩展	2,308 KB
ONSClient4CPP	2016/12/20 10:57	Object File Library	770 KB
Microsoft.AI.PerfCounterCollector.dll	2016/6/6 15:40	应用程序扩展	81 KB
Microsoft.AI.Web.dll	2016/6/6 15:40	应用程序扩展	50 KB
Microsoft.AI.Web	2016/6/6 15:40	XML 文档	27 KB
Microsoft.AI.WindowsServer.dll	2016/6/6 15:40	应用程序扩展	40 KB
Microsoft.AI.WindowsServer	2016/6/6 15:40	XML 文档	26 KB
Microsoft.AI.DependencyCollector.dll	2016/6/6 15:40	应用程序扩展	60 KB
Microsoft.AI.DependencyCollector	2016/6/6 15:40	XML 文档	60 KB
Microsoft.AI.ServerTelemetryChannel....	2016/6/6 10:45	应用程序扩展	89 KB
Microsoft.AI.ServerTelemetryChannel	2016/6/6 10:45	XML 文档	57 KB
Microsoft.ApplicationInsights.dll	2016/6/6 10:42	应用程序扩展	160 KB

ONSClient4CPP 修改日期: 2016/12/20 10:58 创建日期: 2016/12/27 16:28
Program Debug Database 大小: 20.0 MB

1. Choose **Tools** > **Options** > **Projects and Solutions** > **Web Projects**. On the page that is displayed, select **Use the 64 bit version of IIS Express for web sites and projects** .





[✓] Use the 64 bit version of IIS Express for web sites and projects

[ ] Warn before running web applications when there are errors in the error list

Send and receive normal messages

Send normal messages

Execute the following code to send messages. Set parameters correctly according to the instructions.

```
using System;
using ons;

public class ProducerExampleForEx
{
public ProducerExampleForEx()
{
}

static void Main(string[] args) {
// Configure your account according to the settings in the console.
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
// The group ID you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "GID_example");
// The topic you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_name");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
// Set the log path.
factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");

// Create producer instances.
// Note: Producer instances are thread-secure and can be used to send messages of different topics. Each of your
threads
// needs only one producer instance.
Producer producer = ONSFactory.getInstance().createProducer(factoryInfo);

// Start the instance at the client.
producer.start();

// Create message objects.
Message msg = new Message(factoryInfo.getPublishTopics(), "tagA", "Example message body");
msg.setKey(Guid.NewGuid(). ToString());
for (int i = 0; i < 32; i++) {
try
{
SendResultONS sendResult = producer.send(msg);
Console.WriteLine("send success {0}", sendResult.getMessageId());
}
catch (Exception ex)
{
Console.WriteLine("send failure{0}", ex.ToString());
}
}
// Disable the producer instance when the thread is about to exit.
producer.shutdown();

}
}
```

Subscribe to normal messages

For instructions and sample codes of subscribing to normal messages, see Subscribe to messages.

Send and receive ordered messages

Send ordered messages

The sample code for sending ordered messages is as follows:

```

using System;
using ons;

public class OrderProducerExampleForEx
{
public OrderProducerExampleForEx()
{
}

static void Main(string[] args) {
// Configure your account according to the following settings. You can obtain these settings in the console.
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
// The group ID you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.ProducerId, "GID_example");
// The topic you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_name");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
// Set the log path.
factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");

// Create producer instances.
// Note: Producer instances are thread-secure and can be used to send messages of different topics. Each of your
threads
// needs only one producer instance.
OrderProducer producer = ONSFactory.getInstance().createOrderProducer(factoryInfo);

// Start the instance at the client.
producer.start();

// Create message objects.
Message msg = new Message(factoryInfo.getPublishTopics(), "tagA", "Example message body");
string shardingKey = "App-Test";
for (int i = 0; i < 32; i++) {
try
{
SendResultONS sendResult = producer.send(msg, shardingKey);
Console.WriteLine("send success {0}", sendResult.getMessageId());
}
catch (Exception ex)
{
Console.WriteLine("send failure{0}", ex.ToString());
}
}

// Disable the producer instance when the thread is about to exit.
producer.shutdown();

}
}
```

The sample code for consuming ordered messages is as follows:

```


using System;
using System.Text;
using System.Threading;
using ons;

namespace demo
{

public class MyMsgOrderListener : MessageOrderListener
{
public MyMsgOrderListener()
{

}

~MyMsgOrderListener()
{
}

public override ons.OrderAction consume(Message value, ConsumeOrderContext context)
{
Byte[] text = Encoding.Default.GetBytes(value.getBody());
Console.WriteLine(Encoding.UTF8. GetString(text));
return ons.OrderAction.Success;
}
}
class OrderConsumerExampleForEx
{
static void Main(string[] args)
{
// Configure your account according to the following settings. You can obtain these settings in the console.
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
// The group ID you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.ConsumerId, "GID_example");
// The topic you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_name");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
// Set the log path.
factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");

// Create producer instances.
OrderConsumer consumer = ONSFactory.getInstance().createOrderConsumer(factoryInfo);

// Subscribe to topics.
consumer.subscribe(factoryInfo.getPublishTopics(), "*",new MyMsgOrderListener());

// Start the consumer instance.
consumer.start();

// Enable the main thread to sleep for a period of time.
Thread.Sleep(30000);

// Disable the consumer instance when you no longer use it.
consumer.shutdown();
}
}
}
```

Send and receive scheduled messages

The currently supported domains include Internet, East China 1, East China 2, North China 2, and South China 1.

Scheduled messages can be consumed by consumers after a specified period, which are used in scenarios where there are time window requirements for message production and consumption, or when messages are used to trigger scheduled tasks, similar to delayed queues.

Send Scheduled Messages

The following are sample codes for sending scheduled messages:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

namespace ons
{
class onscsharp
{
static void Main(string[] args)
{
//A mandatory parameter required for the producer creation and message handling
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
factoryInfo.setFactoryProperty(factoryInfo.PublishTopics, "XXX");//The topic you created on the MQ console
factoryInfo.setFactoryProperty(factoryInfo.MsgContent, "XXX");//Message content
factoryInfo.setFactoryProperty(factoryInfo.AccessKey, "XXX");//AccessKey, Alibaba Cloud ID verification, which is
created on Alibaba Cloud Management Console
factoryInfo.setFactoryProperty(factoryInfo.SecretKey,"XXX");//SecretKey, Alibaba Cloud ID verification, which is
created on Alibaba Cloud Management Console

//Create a producer
ONSFactory onsfactory = new ONSFactory();
Producer pProducer = onsfactory.getInstance().createProducer(factoryInfo);

//Before sending messages, the start method must be called once to start the producer.
pProducer.start();

Message msg = new Message(
//Message Topic
factoryInfo.getPublishTopics(),
//Message Tag
"TagA",
//Message Body
factoryInfo.getMessageContent()
);

// The setting represents the key service property of the message, so please set it as globally unique as possible.
// You can query a message and resend it through the MQ console when you cannot receive the message properly.
// Note: Normal sessage sending and receiving will not be affected if message key is not configured.
msg.setKey("ORDERID_100");

// Deliver time (ms) specifies the time point after which the message can be consumed. The example means that
the message will be consumed after 3 seconds.
long deliverTime = obtain current system time (ms) + 3000;
msg.setStartDeliverTime(deliverTime);

//If no exceptions are thrown, then the message is sent successfully.
try
{
SendResultONS sendResult = pProducer.send(msg);
}
catch(ONSClientException e)
{
//Handle the message sending failures
}

//The object Producer must be destroyed before exiting the application. Otherwise there will be memory leakage.
pProducer.shutdown();

}
}
}
```

Subsribe to Scheduled Messages

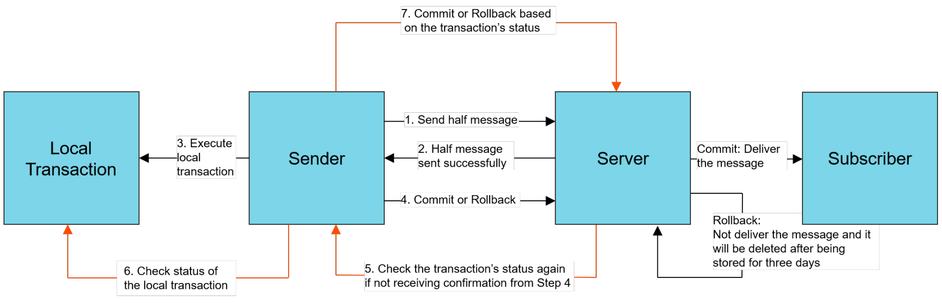
For instructions and example codes of subscribing to scheduled messages, see Subscribe to messages.

Send and receive transactional messages

The currently supported regions include Internet, China (Hangzhou), China (Beijing), China (Shanghai), and China (Shenzhen).

Interaction process

The following figure shows the interaction process of MQ transactional messages.


7. Commit or Rollback based on the transaction's status
Local Transaction
3. Execute local transaction
Sender
1. Send half message
2. Half message sent successfully
4. Commit or Rollback
Server
Commit: Deliver the message
Subscriber
Rollback: Not deliver the message and it will be deleted after being stored for three days
6. Check status of the local transaction
5. Check the transaction's status again if not receiving confirmation from Step 4

Send transactional messages

Follow these steps to send a transactional message:

Send a half message and execute a local transaction. The sample code is as follows:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using ons;

namespace ons
{
public class MyLocalTransactionExecuter : LocalTransactionExecuter
{
public MyLocalTransactionExecuter()
{
}

~MyLocalTransactionExecuter()
{
}
public override TransactionStatus execute(Message value)
{
Console.WriteLine("execute topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4}, userProperty:{5}",
value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.getBody(),
value.getUserProperty("VincentNoUser"));

//The ID of the message. Two messages can have the same message body but different message IDs.
Currently, message IDs cannot be queried in the console.
string msgId = value.getMsgID();
// Compute the message body by using CRC32, MD5, or other algorithms.
// The message ID and CRC32 ID are used to prevent duplication of messages.
// To avoid duplication of messages, compute the message body by using the CRC32 or MD5 algorithm.

TransactionStatus transactionStatus = TransactionStatus.Unknow;
try {
boolean isCommit = Execution result of the local transaction;
if (isCommit) {
// Submit the message if the local transaction succeeds.
transactionStatus = TransactionStatus.CommitTransaction;
} else {
// Roll back the message if the local transaction fails.
transactionStatus = TransactionStatus.RollbackTransaction;
}
} catch (Exception e) {
//exception handle
}
return transactionStatus;
}
}
class onscsharp
{

static void Main(string[] args)
{
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
factoryInfo.setFactoryProperty(factoryInfo.NAMESRV_ADDR, "XXX");//Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint Information** area.
factoryInfo.setFactoryProperty(factoryInfo.ProducerId, "");//The group ID you created in the console.
factoryInfo.setFactoryProperty(factoryInfo.PublishTopics, "");//The topic you created in the console.
factoryInfo.setFactoryProperty(factoryInfo.MsgContent, "");//The message body.
factoryInfo.setFactoryProperty(factoryInfo.AccessKey, "");//The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(factoryInfo.SecretKey, "");//The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.

//Create transaction producers
ONSFactory onsfactory = new ONSFactory();
LocalTransactionChecker myChecker = new MyLocalTransactionChecker();
TransactionProducer pProducer = onsfactory.getInstance().createTransactionProducer(factoryInfo,ref myChecker);

// Before sending messages, call the start method once to start the producer. After the producer is started, messages can be concurrently sent through multiple threads.
pProducer.start();

Message msg = new Message(
//The message topic.
factoryInfo.getPublishTopics(),
//The message tag.
"TagA",
// The message body.
factoryInfo.getMessageContent()
);

// Set a key service property representing the message, that is, the message key, and try to keep it globally unique.
// A unique identifier enables you to query a message and resend it in the console if you fail to receive the message.
// Note: Messages can still be sent and received if you do not set this attribute.
msg.setKey("ORDERID_100");

// The message is sent if no exception is thrown.
try
{
LocalTransactionExecuter myExecuter = new MyLocalTransactionExecuter();
SendResultONS sendResult = pProducer.send(msg, ref myExecuter);
}
catch(ONSClientException e)
{
Console.WriteLine("\nexception of sendmsg:{0}",e.what() );
}

// Destroy the producer before exiting the application. Otherwise, memory leakage may occur.
// The producer cannot be started again after shutdown.
pProducer.shutdown();
}
}
}
```

Submit the transactional message status.

After the execution of a local transaction (successful or failed), the broker must be notified of the transaction status of the current message. Two notification modes are supported:

- Submit the status after executing the local transaction.
- Wait until the broker requests to check the transaction status of the message.

A transaction may be in one of the following states:

- TransactionStatus.CommitTransaction: The transaction is submitted, and the consumer can consume the message.
- TransactionStatus.RollbackTransaction: The transaction is rolled back, and the message is discarded and cannot be consumed.
- TransactionStatus.Unknow: The transaction is in an unknown status, and the broker is expected to query the status of the local transaction that corresponds to the message from the message sender.

```
 Public class MyLocalTransactionChecker: LocalTransactionChecker
{
public MyLocalTransactionChecker()
{
}

~MyLocalTransactionChecker()
{
}

public override TransactionStatus check(Message value)
{
Console.WriteLine("check topic: {0}, tag:{1}, key:{2}, msgId:{3},msgbody:{4}, userProperty:{5}",
value.getTopic(), value.getTag(), value.getKey(), value.getMsgID(), value.getBody(),
value.getUserProperty("VincentNoUser"));
// The ID of the message. Two messages can have the same message body but different message IDs.
Currently, message IDs cannot be queried in the console.
string msgId = value.getMsgID();
// Compute the message body by using CRC32, MD5, or other algorithms.
// The message ID and CRC32 ID are used to prevent duplication of messages.
// You do not need to specify the message ID or CRC32 ID if the business itself achieves idempotence.
Otherwise, set the message ID or CRC32 ID to ensure idempotence.
// To avoid duplication of messages, compute the message body by using the CRC32 or MD5 algorithm.
TransactionStatus transactionStatus = TransactionStatus.Unknow;
try {
boolean isCommit = Execution result of the local transaction;
if (isCommit) {
// If the local transaction succeeded, the message is submitted.
transactionStatus = TransactionStatus.CommitTransaction;
} else {
// If the local transaction failed, the message is rolled back.
transactionStatus = TransactionStatus.RollbackTransaction;
}
} catch (Exception e) {
//exception handle
}
return transactionStatus;
}
}
```

Transaction check mechanism

Why must the transaction status check mechanism be implemented when transactional messages are sent?

When a half message is sent in step 1, but either the returned status of the local transaction is TransactionStatus.Unknow, or no status is submitted because the application exits, the status of the half message is unknown to the MQ broker. Therefore, the MQ broker requires the message sender to periodically check the status of the half message and report the final status.

What does the business logic do when the check method is called back?

The check method for transactional messages needs to contain the logic of transaction consistency check. After a transactional message is sent, MQ needs to use the LocalTransactionChecker operation to respond to the request of the broker for the local transaction status. Therefore, the check method for the transactional message needs to complete the following tasks:

(1) Check the status of the local transaction corresponding to the half message (committed or rollback).

(2) Submit the status of the local transaction to the broker.

What is the impact of different local transaction statuses on the half message?

TransactionStatus.CommitTransaction: The transaction is submitted, and the consumer can consume the message.

TransactionStatus.RollbackTransaction: The transaction is rolled back, and the message is discarded and cannot be consumed.

TransactionStatus.Unknow: The transaction is in an unknown status, and the broker is expected to query the status of the local transaction that corresponds to the message from the message sender.

For more information about the code, see the implementation of MyLocalTransactionChecker.

Subscribe to transactional messages

For instructions and sample codes of subscribing to normal messages, see Subscribe to messages.

Subscribe to messages

This topic describes how to subscribe to messages by using the .NET SDK of MQ.

Note:

Maintain consistent subscription for all consumer instances with the same group ID. For more information, see Subscription consistency.

Subscription modes

MQ supports the following two message subscription modes:

Clustering subscription: All the consumers identified by the same group ID equally share messages. For example, a topic contains nine messages and a group contains three consumer instances. In this case, each instance consumes three messages.

```
 // The configuration of clustering subscription (default mode).
factoryInfo.setFactoryProperty(ONSFactoryProperty.MessageModel, ONSFactoryProperty.CLUSTERING);
```

Broadcasting subscription: All the consumers identified by the same group ID consume every message once. For example, a topic contains nine messages and a group contains three consumer instances. In this case, each instance consumes nine messages.

```
 // The configuration of broadcasting subscription.
factoryInfo.setFactoryProperty(ONSFactoryProperty.MessageModel,
ONSFactoryProperty.BROADCASTING);
```

Sample code

```
using System;
using System.Threading;
using System.Text;
using ons;

// The callback function you need to execute when the message is pulled from the broker.
public class MyMsgListener : MessageListener
{
public MyMsgListener()
{
}

~MyMsgListener()
{
}

public override ons.Action consume(Message value, ConsumeContext context)
{
Byte[] text = Encoding.Default.GetBytes(value.getBody());
Console.WriteLine(Encoding.UTF8. GetString(text));
return ons.Action.CommitMessage;
}
}

public class ConsumerExampleForEx
{
public ConsumerExampleForEx()
{
}

static void Main(string[] args) {
// Configure your account according to the following settings. You can obtain these settings in the console.
ONSFactoryProperty factoryInfo = new ONSFactoryProperty();
// The AccessKeyId you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.AccessKey, "Your access key");
// The AccessKeySecret you created in the Alibaba Cloud console for identity authentication.
factoryInfo.setFactoryProperty(ONSFactoryProperty.SecretKey, "Your access secret");
// The group ID you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.ConsumerId, "GID_example");
// The topic you created in the console.
factoryInfo.setFactoryProperty(ONSFactoryProperty.PublishTopics, "T_example_topic_name");
// Set the TCP endpoint: Go to the **Instances** page in the MQ console, and view the endpoint in the **Endpoint
Information** area.
factoryInfo.setFactoryProperty(ONSFactoryProperty.NAMESRV_ADDR, "NameSrv_Addr");
// Set the log path.
factoryInfo.setFactoryProperty(ONSFactoryProperty.LogPath, "C://log");
// Clustering consumption
// factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty.CLUSTERING);
// Broadcasting consumption
// factoryInfo.setFactoryProperty(ONSFactoryProperty:: MessageModel, ONSFactoryProperty.BROADCASTING);

// Create consumer instances.
PushConsumer consumer = ONSFactory.getInstance().createPushConsumer(factoryInfo);

// Subscribe to topics.
consumer.subscribe(factoryInfo.getPublishTopics(), "*", new MyMsgListener());

// Start the instance at the client.
consumer.start();

//This setting is only used in the demo. In actual production environment, you cannot exit the process.
Thread.Sleep(300000);

// Disable the consumer instance when the process is about to exit.
consumer.shutdown();
}
}
```

SDK guide (HTTP)

SDK guide (HTTP)

HTTP-based MQ instances are currently supported in the following regions:

- China (Hangzhou)
- China (Shanghai)
- China (Shenzhen)
- China (Beijing)
- Germany (Frankfurt)

This type of instance will soon be supported in other regions.

Supported languages

MQ supports HTTP communication based on RESTful and provides SDKs in the following seven languages:

- Go
- Python
- Node.js
- PHP
- Java

- C++
- C#

SDKs and sample codes

Go to MQ HTTP SDK Repository to download the SDKs of the required languages, to read SDK instructions, and to view sample codes for sending and receiving messages.

Limits

HTTP-based MQ instances have the following limits:

- Currently, advanced feature messages are not supported, including ordered messages, transactional messages, and scheduled messages.
- Currently, message trace query is not supported.
- Currently, broadcasting mode is not supported.
- The group IDs of TCP-based instances cannot be used for HTTP-based instances and vice versa. You need to create a group ID for a TCP-based instance and an HTTP-based instance separately.