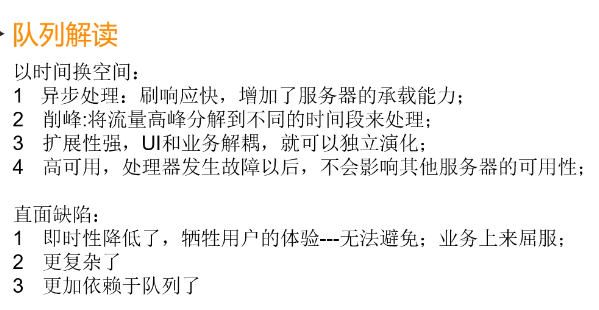
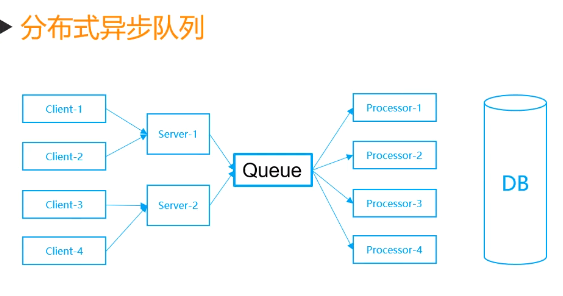
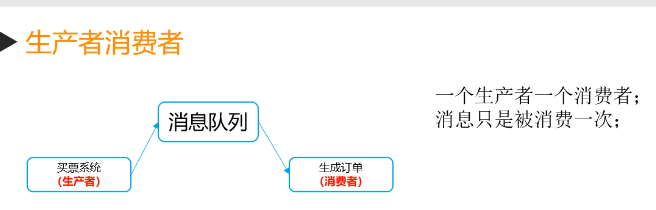
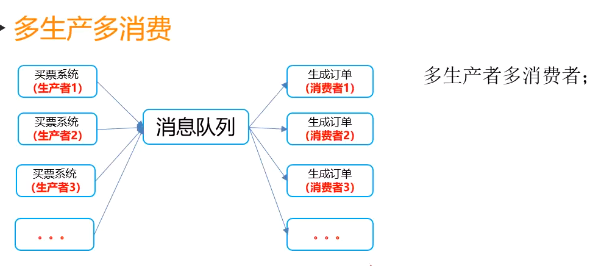
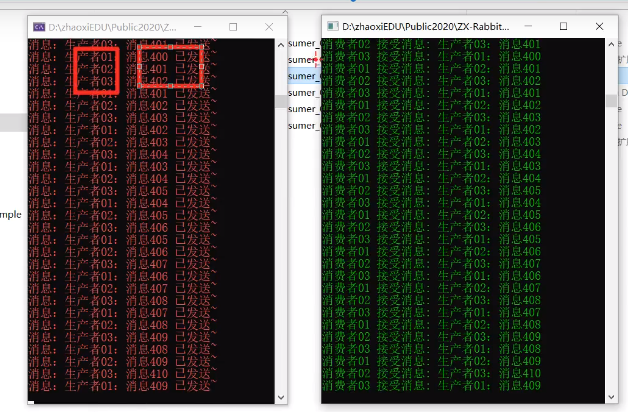
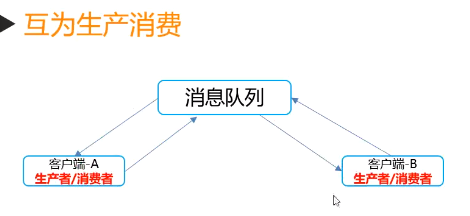
## 定义

消息总线(Message Queue)，是一种跨进程、异步的通信机制，用于上下游传递消息。**由消息系统来确保消息的可靠传递。**

场景：应用解耦、异步、流量削锋、数据分发、错峰流控、日志收集等等...



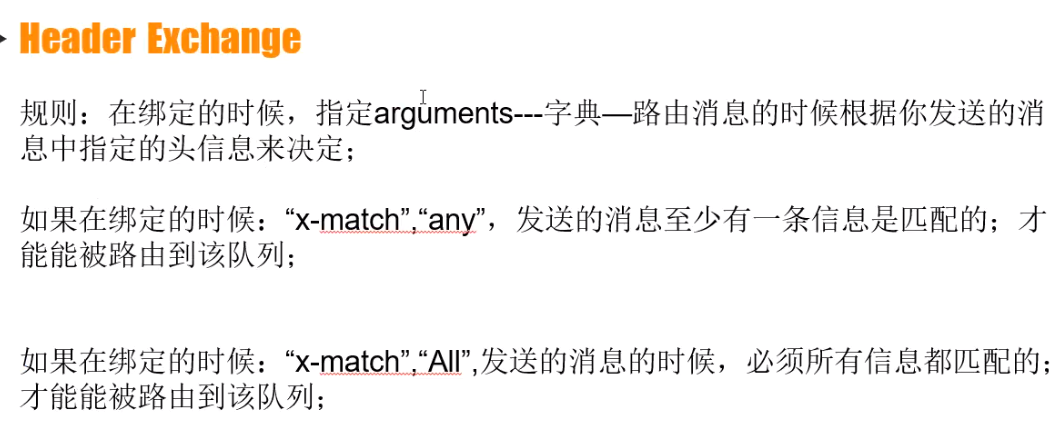
## Exchange 4种类型

1.Direct Exchange(指定路由): 消息中的路由键(routing key)如果和Binding中的binding key一致，交换器就将消息发到对应的队列中。路由键与队列名完全匹配。

2.Fanout Exchange(发布订阅)没有路由键，只要绑定这个exchange，所有队列都可以接受到消息

3.Topic Exchange(占位符指定路由)：exchange需要指定key，key有自己的规则：key可以有占位符 \*(匹配一个词) #(匹配多个词)

4.Header Exchange



如何保证RabbitMQ消息的ACID

把生产端消息确认和消费者消息确认结合起来就行

## 消息优先级：

var pops = channel.CreateBasicProperties();

channel.BasicPublish(exchange: "PriorityQueueExchange", routingKey: "PriorityKey", basicProperties: pops, body: Encoding.UTF8.GetBytes(“wwe消息”));

## 生产端消息确认

1. TX事务模式:保证所有消息同时发送成功，否则同时失败

//开启事务机制

channel.TxSelect();

//事务提交

channel.TxCommit();

//失败-回滚消息

channel. TxRollback ();

1. Confirm模式:

//开启消息确认机制

channel.ConfirmSelect();

//消息发送回执方法， 如果一个消息或者多个消息都确认发送,

if (channel.WaitForConfirms())

{

}

channel.WaitForConfirmsOrDie(); //类似tx事务，全成功正常执行，有失败的，就抛出异常

## 消费端消息确认

//手动确认 消息正常消费，告诉broke, 这个消息可以删除了

channel.BasicAck(deliveryTag: ea.DeliveryTag, multiple: false);

//否定，告诉broke，消息异常，requeue：true(重新写入队列) false（删除这个消息）

channel.BasicReject(deliveryTag: ea.DeliveryTag, requeue: true);

//autoAck: true(自动确认消息完成，可以删除) false(显示确认)

channel.BasicConsume(queue: "OnlyProducerMessage", autoAck: true, consumer: consumer);

## 代码

先添加RabbitMQ.Client包

### 调用

//创建单个生产者

// RabbitMQProductionConsumer.CreateSingleProducer();

//生产者接收到请求时Received事件

//RabbitMQProductionConsumer.ReveicedProducerEvent();

//多个生产者,多个线程同时开始

//生产者01

// Task.Run(() => { RabbitMQProductionConsumer.CreateMutiProducer(1); });

//生产者02

//Task.Run(() => { RabbitMQProductionConsumer.CreateMutiProducer(2); });

//DirectExchange

//Exchange.DirectExchangeProducer();

//Exchange.DirectExchangeConsumerLogAll();

//Exchange.DirectExchangeConsumerLogError();

//FanoutExchange

//Exchange.FanoutExchangeProducer();

//Exchange.FanoutExchangeConsumer("Consumer001");

//Exchange.FanoutExchangeConsumer("Consumer002");

//Priority 优先级

//PriorityQueue.PriorityProducer();

//key要对应上

//RabbitMQProductionConsumer.ReveicedProducerEventByConsumer();

//消息确认 2种方式: 1. Tx事务模式 2. Confirm模式:

//MessageAffirm.MessageTx();

//MessageAffirm.MessageComfirm();

### 帮助类

using RabbitMQ.Client;

using RabbitMQ.Client.Events;

using System;

using System.Collections.Generic;

using System.Text;

using System.Threading;

namespace Zhaoxi.Helper

{

/// <summary>

/// RabbitMQ 操作类

/// </summary>

public class RabbitMQProductionConsumer

{

/// <summary>

/// 创建多个生产者

/// </summary>

/// <param name="num">第几个生产者</param>

public static void CreateMutiProducer(int num = 1)

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

channel.QueueDeclare(queue: "MutiProducerMessage", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "MutiProducerMessageExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "MutiProducerMessage", exchange: "MutiProducerMessageExchange", routingKey: string.Empty, arguments: null);

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("生产者001准备就绪");

int i = 1;

while (true)

{

//多个生产者，这里的生产者01就是生产者n

string message = $"生产者{num} 消息{i}";

byte[] body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "MutiProducerMessageExchange", routingKey: string.Empty, basicProperties: null, body: body);

Console.WriteLine($"消息{message}已发送");

i++;

Thread.Sleep(200);

}

}

}

}

/// <summary>

/// （消费者）生产者接收到请求时Received事件

/// </summary>

public static void ReveicedProducerEventByConsumer()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

Console.ForegroundColor = ConsoleColor.Green;

try

{

var consumer = new EventingBasicConsumer(channel);

consumer.Received += (model, ea) =>

{

//如果有消息需要处理，就触发这个事件

var body = ea.Body;

var message = Encoding.UTF8.GetString(body.ToArray());

Console.WriteLine($"消费者01接受的消息是{message}");

};

channel.BasicConsume(queue: "OnlyProducerMessage", autoAck: true, consumer: consumer);

Console.WriteLine($"Press [enter] to exit");

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

/// <summary>

/// 创建单个生产者

/// </summary>

public static void CreateSingleProducer()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

channel.QueueDeclare(queue: "OnlyProducerMessage", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "OnlyProducerMessageExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "OnlyProducerMessage", exchange: "OnlyProducerMessageExchange", routingKey: string.Empty, arguments: null);

Console.ForegroundColor = ConsoleColor.Red;

Console.WriteLine("生产者ProductionDemo准备就绪");

int i = 1;

while (true)

{

string message = $"消息{i}";

byte[] body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "OnlyProducerMessageExchange", routingKey: string.Empty, basicProperties: null, body: body);

Console.WriteLine($"消息{message}已发送");

i++;

Thread.Sleep(200);

}

}

}

}

}

/// <summary>

/// 路由

/// </summary>

public class Exchange

{

#region DirectExchange

/// <summary>

/// DirectExchange生产者

/// </summary>

public static void DirectExchangeProducer()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

//生产日志写入rabitmq中

channel.QueueDeclare(queue: "DirectExchangeLogAllQueue", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.QueueDeclare(queue: "DirectExchangeErrorQueue", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "DirectExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

string[] logtypes = new string[] { "debug", "info", "warn", "error" };

foreach (var item in logtypes)

{

channel.QueueBind(queue: "DirectExchangeLogAllQueue", exchange: "DirectExchange", routingKey: item);

}

channel.QueueBind(queue: "DirectExchangeErrorQueue", exchange: "DirectExchange", routingKey: "error");

List<LogMsgModel> loglist = new List<LogMsgModel>();

for (int i = 0; i < 100; i++)

{

if (i % 4 == 0)

{

loglist.Add(new LogMsgModel() { LogType = "info", Msg = Encoding.UTF8.GetBytes($"info第{i}条信息") });

}

if (i % 4 == 1)

{

loglist.Add(new LogMsgModel() { LogType = "debug", Msg = Encoding.UTF8.GetBytes($"debug第{i}条信息") });

}

if (i % 4 == 2)

{

loglist.Add(new LogMsgModel() { LogType = "warn", Msg = Encoding.UTF8.GetBytes($"warn第{i}条信息") });

}

if (i % 4 == 3)

{

loglist.Add(new LogMsgModel() { LogType = "error", Msg = Encoding.UTF8.GetBytes($"error第{i}条信息") });

}

}

Console.WriteLine("生产者发送100条信息");

foreach (var item in loglist)

{

channel.BasicPublish(exchange: "DirectExchange", routingKey: item.LogType, basicProperties: null, body: item.Msg);

Console.WriteLine($"{Encoding.UTF8.GetString(item.Msg)} 已发送~");

}

}

}

}

/// <summary>

/// DirectExchange消费者 所有日志

/// </summary>

public static void DirectExchangeConsumerLogAll()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

//生产日志写入rabitmq中

channel.QueueDeclare(queue: "DirectExchangeLogAllQueue", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "DirectExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

string[] logtypes = new string[] { "debug", "info", "warn", "error" };

foreach (var item in logtypes)

{

channel.QueueBind(queue: "DirectExchangeLogAllQueue", exchange: "DirectExchange", routingKey: item);

}

//消费队列中的所有消息

var consumer = new EventingBasicConsumer(channel);

consumer.Received += (model, ea) =>

{

//如果有消息需要处理，就触发这个事件

var body = ea.Body;

var message = Encoding.UTF8.GetString(body.ToArray());

Console.WriteLine($"【{message}】,写入文本");

};

channel.BasicConsume(queue: "DirectExchangeLogAllQueue", autoAck: true, consumer: consumer);

Console.ReadKey();

}

}

}

/// <summary>

/// DirectExchange消费者 错误日志

/// </summary>

public static void DirectExchangeConsumerLogError()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

//生产日志写入rabitmq中,队列名重复，会使用之前的队列

channel.QueueDeclare(queue: "DirectExchangeErrorQueue", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "DirectExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

string[] logtypes = new string[] { "debug", "info", "warn", "error" };

foreach (var item in logtypes)

{

channel.QueueBind(queue: "DirectExchangeErrorQueue", exchange: "DirectExchange", routingKey: item);

}

//消费队列中的所有消息

var consumer = new EventingBasicConsumer(channel);

consumer.Received += (model, ea) =>

{

//如果有消息需要处理，就触发这个事件

var body = ea.Body;

var message = Encoding.UTF8.GetString(body.ToArray());

Console.WriteLine($"【{message}】,已经发送邮件通知管理员!");

};

channel.BasicConsume(queue: "DirectExchangeErrorQueue", autoAck: true, consumer: consumer);

Console.ReadKey();

}

}

}

#endregion

#region FanoutExchange

/// <summary>

/// FanoutExchange生产者

/// </summary>

public static void FanoutExchangeProducer()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

//生产日志写入rabitmq中

channel.QueueDeclare(queue: "FanoutExchange01", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.QueueDeclare(queue: "FanoutExchange02", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "FanoutExchange", type: ExchangeType.Fanout, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "FanoutExchange01", exchange: "FanoutExchange", routingKey: string.Empty, arguments: null);

channel.QueueBind(queue: "FanoutExchange02", exchange: "FanoutExchange", routingKey: string.Empty, arguments: null);

int i = 1;

while (true)

{

var message = $"通知{i}";

var body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "FanoutExchange", routingKey: string.Empty, basicProperties: null, body: body);

Console.WriteLine($"通知【{message}】已经发送到消息队列");

Thread.Sleep(2000);

i++;

}

}

}

}

/// <summary>

/// FanoutExchange消费者，多个

/// </summary>

/// <param name="consumerName">消费者名称</param>

public static void FanoutExchangeConsumer(string consumerName)

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

//生产日志写入rabitmq中

channel.QueueDeclare(queue: "FanoutExchange" + consumerName, durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "FanoutExchange", type: ExchangeType.Fanout, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "FanoutExchange" + consumerName, exchange: "FanoutExchange", routingKey: string.Empty, arguments: null);

//消费队列中的所有消息

var consumer = new EventingBasicConsumer(channel);

consumer.Received += (model, ea) =>

{

//如果有消息需要处理，就触发这个事件

var body = ea.Body;

var message = Encoding.UTF8.GetString(body.ToArray());

Console.WriteLine($"【{message}】,写入文本");

};

channel.BasicConsume(queue: "FanoutExchange" + consumerName, autoAck: true, consumer: consumer);

Console.ReadKey();

}

}

}

#endregion

#region TopicExchange

/// <summary>

/// TopicExchange生产者

/// </summary>

public static void TopicExchangeProducer()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

channel.QueueDeclare(queue: "ChinaQueue", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.QueueDeclare(queue: "NewsQueue", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "TopicExchange", type: ExchangeType.Topic, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "ChinaQueue", exchange: "TopicExchange", routingKey: "China.#", arguments: null);

channel.QueueBind(queue: "NewsQueue", exchange: "TopicExchange", routingKey: "#.News", arguments: null);

//消费者

{

//匹配队列 ChinaQueue NewsQueue

string message = "News from china ";

var body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "TopicExchange", routingKey: "China.News", basicProperties: null, body: body);

Console.WriteLine($"消息{message}已发送到消息队列");

}

{

//匹配队列 ChinaQueue

string message = "Weather from china ";

var body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "TopicExchange", routingKey: "China.Weather", basicProperties: null, body: body);

Console.WriteLine($"消息{message}已发送到消息队列");

}

{

//匹配队列 NewsQueue

string message = "News from America ";

var body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "TopicExchange", routingKey: "usa.News", basicProperties: null, body: body);

Console.WriteLine($"消息{message}已发送到消息队列");

}

{

//匹配队列 没有匹配

string message = "Weather from America ";

var body = Encoding.UTF8.GetBytes(message);

channel.BasicPublish(exchange: "TopicExchange", routingKey: "usa.Weather", basicProperties: null, body: body);

Console.WriteLine($"消息{message}已发送到消息队列");

}

}

}

}

#endregion

}

/// <summary>

/// 优先级

/// </summary>

public class PriorityQueue

{

public static void PriorityProducer()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

//arguments有参数

channel.QueueDeclare(queue: "PriorityQueue", durable: true, exclusive: false, autoDelete: false, arguments: new Dictionary<string, object>() { { "x-max-priority", 10 } });

//使用ExchangeType.Direct

channel.ExchangeDeclare(exchange: "PriorityQueueExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

//routingKey: "PriorityKey"

channel.QueueBind(queue: "PriorityQueue", exchange: "PriorityQueueExchange", routingKey: "PriorityKey", arguments: null);

string[] questionlist = { "vip学员01问题", "甲同学问题", "乙同学问题", "vip学员02问题" };

var pops = channel.CreateBasicProperties();

foreach (var item in questionlist)

{

if (item.Contains("vip"))

{

pops.Priority = 9;

}

else

{

pops.Priority = 1;

}

channel.BasicPublish(exchange: "PriorityQueueExchange", routingKey: "PriorityKey", basicProperties: pops, body: Encoding.UTF8.GetBytes(item));

Console.WriteLine($"{item}已发送");

}

Console.Read();

}

}

}

}

/// <summary>

/// 消息确认-生产者 2种方式: 1. Tx事务模式 2. Confirm模式

/// </summary>

public class MessageAffirm

{

/// <summary>

/// 1. Tx事务模式

/// </summary>

public static void MessageTx()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

channel.QueueDeclare(queue: "MessageTx001", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.QueueDeclare(queue: "MessageTx002", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "MessageTxExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "MessageTx001", exchange: "MessageTxExchange", routingKey: "MessageTxKey01", arguments: null);

channel.QueueBind(queue: "MessageTx002", exchange: "MessageTxExchange", routingKey: "MessageTxKey02", arguments: null);

string message = "";

while (!message.Equals("quit", StringComparison.CurrentCultureIgnoreCase))

{

message = Console.ReadLine();

var body = Encoding.UTF8.GetBytes(message);

try

{

//开启事务机制

channel.TxSelect();

//发送消息

channel.BasicPublish(exchange: "MessageTxExchange", routingKey: "MessageTxKey01", basicProperties: null, body: body);

channel.BasicPublish(exchange: "MessageTxExchange", routingKey: "MessageTxKey02", basicProperties: null, body: body);

//事务提交

channel.TxCommit();

Console.WriteLine($"【{message}】发送到broke成功");

}

catch (Exception ex)

{

Console.WriteLine($"【{message}】发送到broke失败");

//回滚消息

channel.TxRollback();

}

}

Console.Read();

}

}

}

/// <summary>

/// 2. Confirm模式

/// </summary>

public static void MessageComfirm()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

channel.QueueDeclare(queue: "Comfirm", durable: true, exclusive: false, autoDelete: false, arguments: null);

channel.ExchangeDeclare(exchange: "ComfirmExchange", type: ExchangeType.Direct, durable: true, autoDelete: false, arguments: null);

channel.QueueBind(queue: "Comfirm", exchange: "ComfirmExchange", routingKey: "ComfirmKey", arguments: null);

string message = "";

while (!message.Equals("quit", StringComparison.CurrentCultureIgnoreCase))

{

message = Console.ReadLine();

var body = Encoding.UTF8.GetBytes(message);

try

{

//开启消息确认机制

channel.ConfirmSelect();

//发送消息

channel.BasicPublish(exchange: "ComfirmExchange", routingKey: "ComfirmKey", basicProperties: null, body: body);

//如果一个消息或者多个消息都确认发送

if (channel.WaitForConfirms())

{

Console.WriteLine($"【{message}】发送到broke成功");

}

else

{

//消息发送失败， 重新写入消息

}

channel.WaitForConfirmsOrDie(); //类似tx事务，全成功正常执行，有失败的，就抛出异常

}

catch (Exception ex)

{

Console.WriteLine($"【{message}】发送到broke失败");

}

}

Console.Read();

}

}

}

}

/// <summary>

/// 消息确认-消费者

/// </summary>

public class ConsumptionACKConfirm

{

public static void Show()

{

var factory = new ConnectionFactory();

factory.HostName = "http://localhost";

factory.UserName = "guest";

factory.Password = "guest";

using (var connection = factory.CreateConnection())

{

using (IModel channel = connection.CreateModel())

{

try

{

var consumer = new EventingBasicConsumer(channel);

int i = 0;

consumer.Received += (model, ea) =>

{

//如果有消息需要处理，就触发这个事件

var body = ea.Body;

var message = Encoding.UTF8.GetString(body.ToArray());

if (i <= 50)

{

//手动确认 消息正常消费，告诉broke, 这个消息可以删除了

channel.BasicAck(deliveryTag: ea.DeliveryTag, multiple: false);

}

else

{

//否定，告诉broke，消息异常，requeue：true(重新写入队列) false（删除这个消息）

channel.BasicReject(deliveryTag: ea.DeliveryTag, requeue: true);

}

i++;

};

//autoAck: true(自动确认消息完成，可以删除) false(显示确认)

channel.BasicConsume(queue: "OnlyProducerMessage", autoAck: true, consumer: consumer);

Console.WriteLine($"Press [enter] to exit");

Console.ReadLine();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

}

}

}

}

public class LogMsgModel

{

public string LogType { get; set; }

public byte[] Msg { get; set; }

}