**RabbitMQ客户端第二版使用说明文档**

# 概述

针对RabbitMQ客户端第一版在使用中遇到的问题，我们开发了第二版，此版本解决了第一版的已知问题，并添加了新的功能。此版本目前支持.NET Framework 4.5及更高。

此版本解决了如下问题：

1. connection和channel的复用。

2. 由于服务器或者网络问题而导致的连接中断，客户端在自动重连的过程中，所有的发布和消费操作引发的异常在回调方法中输出。

3. 对RabbitMQ原生的异常进行了封装，让使用者更加容易的知道发生了什么异常，以便进行相应的处理。

4. 提供日志接口，让使用者可以对客户端内部的一些逻辑进行追踪，方便问题排查。

5. 简化了接口的调用方式。

此版本添加的新功能：

1. 发布时设置消息的有效期。

2. 消费时设置预取消息数。

3. 注册服务

# 迭代记录

|  |  |  |
| --- | --- | --- |
| 日期 | 功能 | 说明 |
| 20160808 | 初始化第二版 | 见概述 |
| 20160927 | 添加消费异常重试控制功能 | 在消费消息发生异常时，控制消息重新投递的次数。两种使用方式：  (1) 在连接字符串中指定重试次数。  例如：retryTimes=2  (2) 在订阅消息时指定重试次数。  例如：x => x. WithRetryTimes(3)  在订阅消息时指定重试次数的优先级高于在连接字符串中进行的指定。 |

# 添加引用

### 需要添加引用的DLL

|  |
| --- |
| EP.Base.RabbitMQClient.dll  RabbitMQ.Client.dll  Autofac.dll  Newtonsoft.Json.dll  目录：$/A4商务产品研发/CommonLib/RabbitMQClient/V2 |

# 相关设置

## **1. 连接字符串设置**

### 参数详解

|  |
| --- |
| host=192.168.1.1;port=5672;virtualHost=MyVirtualHost;username=MyUsername;password=MyPassword;prefetchcount=1;retryTimes=2; publisherConfirms=true; timeout=10 |

**prefetchcount：消费者预取消息数，默认值是1。设置为1平衡多个消费者的消费处理工作。设置为0则不限制预取消息数（不推荐）。**

**retryTimes：消费者处理异常时，消息重新投递的次数。**

**publisherConfirms：是否开启发布确认，即是否开启RabbitMQ成功接收到消息的通知，默认值是false。**

**persistentMessages：持久化消息，默认值是true。**

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost") //设置连接字符串，创建客户端 |

## **2. 消息持久化设置**

### 使用示例

|  |
| --- |
| /// <summary>  /// **设置消息递送模式**  /// true：**持久化消息，默认值**（消息持久化到磁盘，Rabbit服务重启后消息还存在）  /// false：**不持久化消息**（Rabbit服务重启后消息丢失）  /// </summary>  [DeliveryMode(false)]  public class TextMessage  {  public string Text { get; set; }  } |

## **3. RPC请求超时时间设置**

### 使用示例

|  |
| --- |
| /// <summary>  /// **在请求响应模式下，设置请求超时时间**  /// 例如：设置请求超时时间为60秒  /// </summary>  [TimeoutSeconds(60)]  public class RequestMessage  {  public string Text { get; set; }  } |

# 创建客户端

## **1. 创建客户端**

### 方法签名

|  |
| --- |
| /// <summary>  /// 创建客户端实例  /// </summary>  /// <param name="connectionString">  /// 连接字符串，例如：  /// host=192.168.1.1;port=5672;virtualHost=MyVirtualHost;  /// username=MyUsername;password=MyPassword;  /// </param>  /// <returns>  /// 返回客户端实例  /// </returns>  static IBus CreateBus(string connectionString) |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost") //创建客户端  bus.Dispose();//释放客户端连接 |

## **2. 创建客户端并注册服务**

### 方法签名

|  |
| --- |
| /// <param name="registerServices">  /// 注册服务  /// 例如：注册日志服务  /// ClientFactory.CreateBus("host=localhost", x => x.Register<IRabbitLogger>(\_ => myLogger));  /// </param>  static IBus CreateBus(string connectionString, Action<IServiceRegister> registerServices) |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost",  x => x.Register< IRabbitLogger>(\_ => new myLogger())) |

# 发布订阅模式

## **1. 同步发布消息**

### 方法签名

|  |
| --- |
| /// <summary>  /// 同步发布消息  /// </summary>  /// <typeparam name="T">消息的类型</typeparam>  /// <param name="message">要发送的消息实体</param>  /// <param name="exChange">交换机名称</param>  /// <param name="routingKey">路由键</param>  /// <returns>  /// **true 表示消息发布成功，false 表示消息发布失败。**  /// **未开启发布确认时，该参数表示是否发布成功。**  /// **开启发布确认时，该参数可能指示是否发布成功，亦可能指示是否确认成功。**  /// </returns>  bool Publish<T>(T message, string exChange, string routingKey) where T : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  bool b = bus.Publish(new TextMessage { Text = " 123" }, "exchangeone", "one");  Console.WriteLine(b); |

## **2.** **同步发布消息并设置参数**

### 方法签名

|  |
| --- |
| /// <summary>  /// 同步发布消息并设置参数  /// </summary>  /// <typeparam name="T">消息类型</typeparam>  /// <param name="message">消息实体</param>  /// <param name="exChange">交换器</param>  /// <param name="configure">  /// 消息配置，e.g. x => x.WithTopic("\*.brighton").WithPriority(2).WithExpires(10000)  /// 过期时间以毫秒为单位  /// </param>  /// <returns>  /// true 表示消息发布成功，false 表示消息发布失败。  /// 未开启发布确认时，该参数表示是否发布成功。  /// 开启发布确认时，该参数可能指示是否发布成功，亦可能指示是否确认成功。  /// </returns>  bool Publish<T>(T message, string exChange, Action<IPublishConfiguration> configure) where T : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  bool b = bus.Publish(new TextMessage { Text = " 123" }, "exchangeone" ,  x => x.WithTopic(\_routingKey).WithExpires(10000));  Console.WriteLine(b); |

## **3. 异步发布消息**

### 方法签名

|  |
| --- |
| /// <summary>  /// 异步发布消息  /// </summary>  /// <typeparam name="T">消息的类型</typeparam>  /// <param name="message">要发送的消息实体</param>  /// <param name="exChange">交换机名称</param>  /// <param name="routingKey">路由键</param>  /// <param name="**publishConfirm**">  /// **消息发布回调方法**  /// **第一个参数：true 表示消息发布成功，false 表示消息发布失败。**  /// **未开启发布确认时，该参数表示是否发布成功。**  /// **开启发布确认时，该参数可能指示是否发布成功，亦可能指示是否确认成功。**  /// **第二个参数：消息实体**  /// **第三个参数：异常信息**  /// </param>  void PublishAsync<T>(T message, string exChange, string routingKey,  Action<bool, T, Exception> publishConfirm) where T : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  bus.PublishAsync(new TextMessage { Text = " 123" }, "exchangeone", "one",  DefaultPublishConfirm);  public static void DefaultPublishConfirm(bool b, TextMessage message, Exception exception)  {  if (b)  {  }  else  {  Console.WriteLine(exception.Message);  }  } |

## **4. 异步发布消息并设置参数**

### 方法签名

|  |
| --- |
| /// <param name="configure">  /// **消息设置：路由键、有效期（以毫秒为单位）**  /// 例如：x => x.WithTopic("\*.brighton").WithExpires(10000)  /// </param>  void PublishAsync<T>(T message, string exChange, Action<IPublishConfiguration> configure, Action<bool, T, Exception> publishConfirm) where T : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  bus.PublishAsync(new TextMessage { Text = " 123" }, "exchangeone",  x => x.WithTopic("one").WithExpires(10000), DefaultPublishConfirm); |

## **5. 订阅消息**

### 方法签名

|  |
| --- |
| /// <summary>  /// 订阅消息  /// </summary>  /// <param name="queue">订阅的消息队列</param>  /// <param name="**addHandlers**">  /// **根据消息的不同类型，添加相应的处理方法。**  /// **处理消息的方法中请添加异常处理try catch，否则未被处理的异常将导致当前消息被不停的重复投递！**  /// </param>  /// <param name="configure">  /// **订阅配置：预取消息数；处理异常时消息被重新投递的次数；**  /// **例如：x => x.WithPrefetchCount(10).WithRetryTimes(3)**  /// </param>  /// <returns>  /// 返回一个ISubscriptionResult  /// 调用 Dispose 或者 ISubscriptionResult.ConsumerCancellation 取消订阅  /// </returns>  ISubscriptionResult Subscribe(string queue, Action<IReceiveRegistration> addHandlers, Action<ISubscriptionConfiguration> configure); |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  bus.Subscribe(\_queue,  x =>  x.Add<TextMessage>(message => HandleTextMessage(message))  .Add<TextMessage2>(message2 => HandleTextMessage2(message2)),  x => x.WithPrefetchCount(10).WithRetryTimes(3)); |

# 请求响应模式（RPC）

## **1. 同步请求**

### 方法签名

|  |
| --- |
| /// <summary>  /// RPC同步请求  /// </summary>  /// <typeparam name="TRequest">请求消息类型</typeparam>  /// <typeparam name="TResponse">响应消息类型</typeparam>  /// <param name="request">请求的消息</param>  /// <param name="exChange">交换机</param>  /// <param name="routingKey">路由键</param>  /// <param name="replyQueueName">应答队列</param>  /// <param name="handleResponse">  /// 响应处理方法  /// 第一个参数：请求处理是否成功  /// 第二个参数：响应的消息  /// 第三个参数：请求处理失败时的异常信息  /// </param>  void Request<TRequest, TResponse>(TRequest request, string exChange, string routingKey, string replyQueueName, Action<bool, TResponse, Exception> handleResponse)  where TRequest : class  where TResponse : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  RequestMessage request = new RequestMessage();  request.Text = "123";  bus.Request<RequestMessage, ResponseMessage>(request, "exchangeRPC", "two",  "queueResponse", HandleResponse);  public static void HandleResponse(bool b, ResponseMessage response, Exception exception)  {  if (b)  {  }  else  {  Console.WriteLine(exception.Message);  }  } |

## **2. 异步请求**

### 方法签名

|  |
| --- |
| /// <summary>  /// RPC异步请求。  /// 参数与RPC同步请求相同。  /// </summary>  void RequestAsync<TRequest, TResponse>(TRequest request, string exChange, string routingKey, string replyQueueName, Action<bool, TResponse, Exception> handleResponse)  where TRequest : class  where TResponse : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  RequestMessage request = new RequestMessage();  request.Text = "123";  bus.RequestAsync<RequestMessage, ResponseMessage>(request, "exchangeRPC", "two",  "queueResponse", HandleResponse); |

## **3. 响应**

### 方法签名

|  |
| --- |
| /// <summary>  /// RPC响应  /// </summary>  /// <typeparam name="TRequest">请求消息类型</typeparam>  /// <typeparam name="TResponse">响应消息类型</typeparam>  /// <param name="responder">  /// 响应处理方法  /// </param>  /// <param name="queue">队列</param>  /// <param name="configure">  /// 响应配置，例如：x => x.WithPrefetchCount(50).WithRetryTimes(3)  /// </param>  IDisposable Respond<TRequest, TResponse>(Func<TRequest, TResponse> responder,  string queue, Action<IResponderConfiguration> configure = null)  where TRequest : class  where TResponse : class; |

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost");  bus.Respond<RequestMessage, ResponseMessage>(HandleRequestMessage, \_queue, x => x.WithPrefetchCount(10000).WithRetryTimes(3));  while (true)  {  }  static ResponseMessage HandleRequestMessage(RequestMessage request)  {  ResponseMessage response = new ResponseMessage();  response.Text = request.Text + " come from rpcserver";  return response;  } |

# 注册服务

## **1. 日志服务**

### 接口约束

|  |
| --- |
| public interface IRabbitLogger  {  void DebugWrite(string format, params object[] args);  void InfoWrite(string format, params object[] args);  void ErrorWrite(string format, params object[] args);  void ErrorWrite(Exception exception);  } |

### 默认实现

空的方法体。

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost",  x => x.Register< IRabbitLogger>(\_ => new myLogger()))//注册日志服务 |

## **2. 消息拦截器服务**

### 接口约束

|  |
| --- |
| public interface IProduceConsumeInterceptor  {  RawMessage OnProduce(RawMessage rawMessage);  RawMessage OnConsume(RawMessage rawMessage);  } |

### 默认实现

直接返回原始消息。

### 使用示例

|  |
| --- |
| var bus = ClientFactory.CreateBus("host=localhost",  x =>  x.Register<IRabbitLogger>(\_ => new NullLogger())//注册日志服务  //注册消息拦截器服务  .Register<IProduceConsumeInterceptor>(\_ => new DefaultInterceptor())); |

# 注意事项

1. 由于connection、channel是复用的，因此，对一个应用程序而言，建立一个连接（即创建一个Ibus的实例）就够用了。如有特殊需求，需要开启发布确认功能，则此时应该再创建一个Ibus的实例，使用该实例进行需要发布确认的相关操作。

2. 在使用请求响应模式（RPC）时，应确保服务端有足够的处理请求的能力，否则会造成请求超时异常。如果服务端处理请求的速度快，建议尝试着把预取消息数设置的稍大一点。如果服务端处理请求的速度慢，建议把请求超时时间设置的长一点或者多开几个服务端的实例。

3. 在请求响应模式下，建议先开启服务端，再开启客户端，以免造成不必要的请求超时。

4. 目前，在请求响应模式下，同一类型的消息只支持一个请求实例。