RabbitMQ-AMQP-0-9-1模型详解

Vhost

创建方式：

- 命令行工具来创建： rabbitmqctl add_vhost qa1
- 管理控制台创建

Connection

Spring CachingConnectionFactory

默认情况下 单连接的连接工厂。

默认，首先会缓存一个Channel，然后再慢慢增 ，高并发使用场景、channelCacheSize 设置缓存的数量 100

默认的缓存模式是缓存通道。

如果把缓存模式设为CacheMode.CONNECTION，则缓存连接以及连接上创建Channel. connectionCacheSize 属性设置缓存多少个连接

CacheMode.CONNECTION 与 Rabbit Admin (AmqpAdmin) 不兼容，不会自动创建exchange、queues 等

exchange

```
Exchange.DeclareOk exchangeDeclare(String exchange,
String type,
boolean durable,    // 交换器是否持久化  避免重启后，要再次创建。 和消息的持久化没关系。
boolean autoDelete,  // 当没有队列绑定到它时 是否自动删除
boolean internal,   // 是否是 MQ 内部使用的， 我们就不能在客户端中使用。
Map<String, Object> arguments)
```

Queue

- **Name** 应用程序可以选择队列名称，或者要求代理为它们生成一个名称，最长 255 字节 UTF-8字符。如想要Broker为我们生成队列名，可以在声明创建Queue时传入空字符""，在返回值中可以取得生成的队列名。
- **Durable** 是否持久存储，如为false， broker restart就没有了
- **Exclusive** 独占，被一个connection独占使用，当connection 关闭时Queue也被删除
- **Auto-delete** 是否在Queue的最后一个消费者关闭时自动删除Queue
- **Arguments** 可选的被插件和Broker特殊特性使用的参数，如message TTL, queue length limit 等

【注意】以amq.开头的队列名称 是保留给Broker内部使用的，如果用户创建这样的队列会异常。

【注意】队列的持久性，跟消息的持久化也没关系。

Queue 的 TTL TIME TO LIVE

autoDelete 队列空闲一段时间之后再删除。

- policy方式

  ```
  rabbitmqctl set_policy expiry ".*" '{"expires":1800000}' --apply-to queues
  ```

  expiry 策略名称 自定义

  ".*" 作用目标名称的正则表达式

  '{"expires":1800000}' 策略定义 过期时间设置 单位毫秒

  --apply-to queues 应用于哪一类实体

代码中声明队列是指定

```
    Map<String, Object> args = new HashMap<String, Object>();
            args.put("x-expires", 1800000);
            channel.queueDeclare("myqueue", false, false, false, args);

    channel.queueDeclare("queue1", false, false, false, args);
```

Publisher

路由不可达

可能情况：

- 交换没有绑定队列
- 交换没法根据消息的路由key把消息路由到队列。

默认情况： 消息丢弃

可以的处理办法：

- 退回
- 死信队列（备用交换）

退回

void basicPublish(String exchange, String routingKey, boolean mandatory, BasicProperties props, byte[] body)

mandatory：true 强制退回， false 不需退回，直接丢弃。

备用交换

- policy

```
rabbitmqctl set_policy mike "^my-direct$" '{"alternate-exchange":"my-ae"}'
```

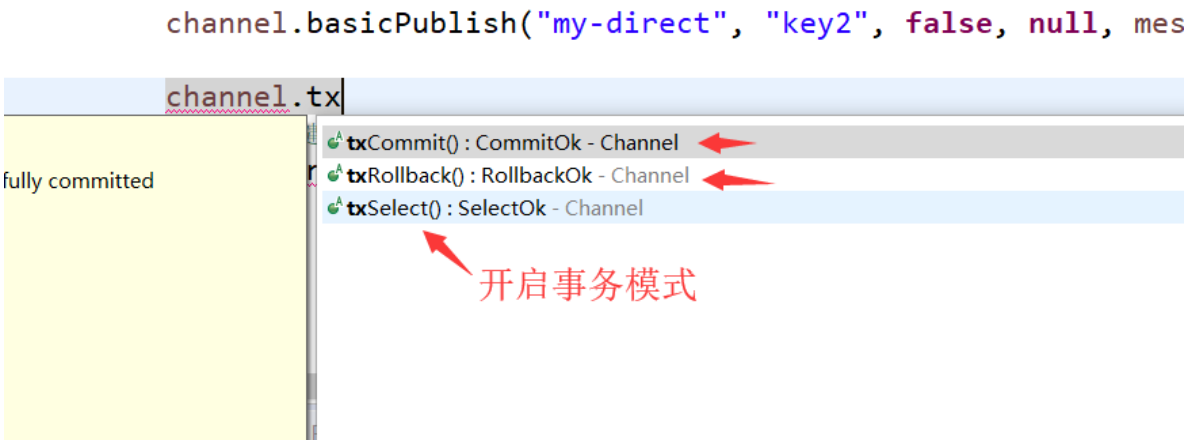
- 代码中声明交换时通过参数指定备用交换

```
//声明参数
Map<String, Object> args = new HashMap<String, Object>();
args.put("alternate-exchange", "my-ae"); //备用交换参数指定
channel.exchangeDeclare("my-direct", "direct", false, false, args);
channel.exchangeDeclare("my-ae", "fanout");
channel.queueDeclare("routed");
channel.queueBind("routed", "my-direct", "key1");
channel.queueDeclare("unrouted");
channel.queueBind("unrouted", "my-ae", "");
```

一定要去动手

事务机制

可靠发布



spring 事务管理需要那些组件?

事务管理器 TransactionManager

```
@Configuration
public class TxConfiguration {
    @Bean       // 配置事务管理器
    public RabbitTransactionManager rabbitTransactionManager(ConnectionFactory
connectionFactory) {
        return new RabbitTransactionManager(connectionFactory);
    }
}
```

在spring 该怎么玩事务就怎么玩.

RabbitTransactionManager 只能做Rabbitmq的消息事务管理 只能是单连接的连接工厂

没有分布式事务管理器实现。

rabbitmq中事务机制来保证消息的可靠发布，性能是比较差

发布确认机制

性能是事务机制的250倍。

发布者发布消息，一般是走异步。

channel

有三种确认模式

- 异步流式确认 事件驱动 开销低，吞吐量大
- 批量发布确认 批次等待，确认不ok 一批重发
- 单条确认 发一条就等待确认

broker给出确认会有三种结果

- ack 接收成功
- nack 接收失败
- 发布者收不到Broker的确认（超时）

异步流式确认

Consumer

两种消息消费模式

- push 推模式

- pull 拉模式

push 模式说明

broker client 消费者

client 向 broker 注册对某个队列的消费者

```
// 对感兴趣的队列注册消费者，返回Server生成的consumerTag（消费者标识）
String consumerTag = channel.basicConsume(queueName, true, callback, consumerTag
-> {});
```

取消消费者注册

```
channel.basicCancel(consumerTag);
```

独占消费者

独占队列：被创建它的连接独占 这个连接上的channel 可以共享。连接关闭，独占队列没有了。

独占消费者：消费者独占一个对队列进行消息消费，适用场景： 消息一定要严格按序消费处理。