AtLeastOnceDelivery

”
3 . .
, .

.
.
.
, .
.

```
namespace AkkaNetCore.Models.Message
{
    // AtLeastOnceDelivery
    public class Msg
    {
        public Msg(long deliveryId, string message)
        {
            DeliveryId = deliveryId;
            Message = message;
        }

        public long DeliveryId { get; }

        public string Message { get; }
    }

    public class Confirm
    {
        public Confirm(long deliveryId)
        {
            DeliveryId = deliveryId;
        }

        public long DeliveryId { get; }
    }

    public interface IEvent
    {

    }

    public class MsgSent : IEvent
    {
        public MsgSent(string message)
        {
            Message = message;
        }

        public string Message { get; }
    }

    public class MsgConfirmed : IEvent
    {
        public MsgConfirmed(long deliveryId)
        {
            DeliveryId = deliveryId;
        }

        public long DeliveryId { get; }
    }
}
```

```
using Akka.Actor;
using Akka.Event;
using Akka.Persistence;
using AkkaNetCore.Models.Message;

namespace AkkaNetCore.Actors.Study
{
    public class CustomActor : ReceiveActor
    {
        private readonly ILoggingAdapter logger = Context.GetLogger();
```

```
    protected int messageCnt = 0;

    public CustomActor()
    {
        Receive<Msg>(msg =>
        {
            messageCnt++;
            if (!msg.Message.Contains(""))
            {
                //  .
                if (messageCnt % 2 == 0)
                {
                    logger.Warning("  .");
                    return;
                }
            }

            //  .
            logger.Debug(", .");
            Sender.Tell(new Confirm(msg.DeliveryId), Self);
        });
    }
}

public class DeliveryManActor : AtLeastOnceDeliveryReceiveActor
{
    private readonly IActorRef _probe;

    private readonly IActorRef _destionationActor;

    private readonly ILoggingAdapter logger = Context.GetLogger();

    public override string PersistenceId { get; } = "persistence-id";

    public DeliveryManActor(IActorRef probe)
    {
        _probe = probe;

        _destionationActor = Context.ActorOf(Props.Create(() => new CustomActor()));

        Recover<MsgSent>(msgSent => Handler(msgSent));

        Recover<MsgConfirmed>(msgConfirmed => Handler(msgConfirmed));

        Command<string>(str =>
        {
            logger.Debug("received:" + str);
            Persist(new MsgSent(str), Handler);
        });

        Command<Confirm>(confirm =>
        {
            //  ,    .
            logger.Debug("received confirm:" + confirm.DeliveryId);
            _probe.Tell("OK");
            Persist(new MsgConfirmed(confirm.DeliveryId), Handler);
        });

    }

    //   -
    private void Handler(MsgSent msgSent)
    {
        logger.Debug(".:" + msgSent.Message);
        Deliver(_destionationActor.Path, l => new Msg(l, msgSent.Message));
    }

    //
    private void Handler(MsgConfirmed msgConfirmed)
    {
```

```
            ConfirmDelivery(msgConfirmed.DeliveryId);
        }
    }
}
```

```
using System;
using Akka.Actor;
using Akka.TestKit;
using AkkaNetCore.Actors.Study;
using Xunit;
using Xunit.Abstractions;

namespace AkkaNetCoreTest.Actors
{
    public class AtLeastOnceDeliveryTest : TestKitXunit
    {
        protected TestProbe probe;

        protected IActorRef deliveryManActor;

        public AtLeastOnceDeliveryTest(ITestOutputHelper output) : base(output)
        {
            Setup();
        }

        public void Setup()
        {
            //     .
            probe = this.CreateTestProbe();

            deliveryManActor = Sys.ActorOf(Props.Create(() => new DeliveryManActor(probe)));

        }

        [Theory]
        [InlineData(3,13000)]
        public void ____(int repeat, int cutoff)
        {
            for (int i = 0; i < repeat; i++)
            {
                deliveryManActor.Tell(":" + i);
            }

            Within(TimeSpan.FromMilliseconds(cutoff), () =>
            {
                for (int i = 0; i < repeat; i++)
                {
                    probe.ExpectMsg<string>(TimeSpan.FromMilliseconds(cutoff));
                }
            });
        }

        [Theory]
        [InlineData(100,300)]
        public void __(int repeat,int cutoff)
        {
            for (int i = 0; i < repeat; i++)
            {
                deliveryManActor.Tell(":" + i);
            }

            Within(TimeSpan.FromMilliseconds(cutoff), () =>
            {
                for (int i = 0; i < repeat; i++)
                {
                    probe.ExpectMsg<string>(TimeSpan.FromMilliseconds(cutoff));
                }
            });
        }

    }
}
```

-____

```
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] received::0
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] .::0
[DEBUG][2020-03-15  1:47:54][Thread 0023][akka://test/user/deliveryman/customer] , .
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] received::1
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] .::1
[WARNING][2020-03-15  1:47:54][Thread 0022][akka://test/user/deliveryman/customer]   .
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] received::2
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] .::2
[DEBUG][2020-03-15  1:47:54][Thread 0022][akka://test/user/deliveryman/customer] , .
[DEBUG][2020-03-15  1:47:54][Thread 0025][akka://test/user/deliveryman] received confirm:1
[DEBUG][2020-03-15  1:47:54][Thread 0024][akka://test/user/deliveryman] received confirm:3
[WARNING][2020-03-15  1:48:01][Thread 0022][akka://test/user/deliveryman/customer]   .
[DEBUG][2020-03-15  1:48:06][Thread 0025][akka://test/user/deliveryman/customer] , .
[DEBUG][2020-03-15  1:48:06][Thread 0022][akka://test/user/deliveryman] received confirm:2
```


Kafka AtleastOnceDeivery ()

, Kafka , Kafka

Erlang .

. (Erlang)

- Erlang 10.9
- Akka-at-least-once-message-delivery
- https://www.joinc.co.kr/w/man/12/Kafka/exactlyonce
- MessageDeliveryReliability -