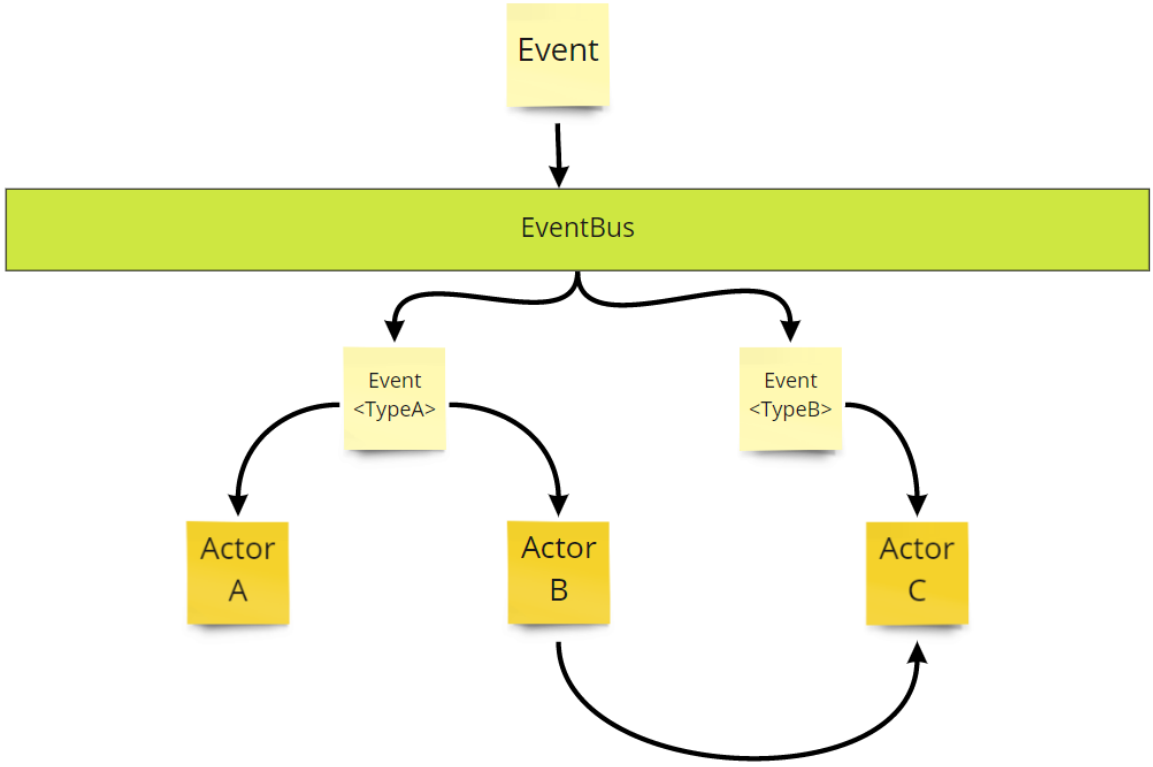
EventBus

,

(Publish)/(SubScribe) .

Akka EventStream EventBus

Actor .



- ActorA ActorB TypeA .
- ActorC TypeB .
- ActorB TypeA ActorC .

Actor , Reply .

Flow , .

- TypeA,TypeB 1 .
- ActorA TypeA 1 .
- ActorB TypeA 1 .
- ActorC TypeB,TypA 1 .

```
        [Theory(DisplayName = " ")]
        [InlineData(1)]
        public void Test1(int testCount)
        {
            var actorA = Sys.ActorOf(Props.Create(() => new ActorA(probeA)));
            var actorB = Sys.ActorOf(Props.Create(() => new ActorB(probeB)));
            var actorC = Sys.ActorOf(Props.Create(() => new ActorC(probeC)));

            // Link ActorB to ActorC
            actorB.Tell(actorC);

            //Subscribe
            Sys.EventStream.Subscribe(actorA, typeof(TypeA));
            Sys.EventStream.Subscribe(actorB, typeof(TypeA));
            Sys.EventStream.Subscribe(actorC, typeof(TypeB));

            //Publish
            Sys.EventStream.Publish(new TypeA() { Text = "TypeA_Message" } );
            Sys.EventStream.Publish(new TypeB() { Text = "TypeB_Message" });

            //UnitTest
            probeA.ExpectMsg<TypeA>(TimeSpan.FromSeconds(1));
            probeB.ExpectMsg<TypeA>(TimeSpan.FromSeconds(1));
            probeC.ExpectMsg<TypeB>(TimeSpan.FromSeconds(1));
            probeC.ExpectMsg<TypeA>(TimeSpan.FromSeconds(1));
        }
```

git : https://github.com/psmon/AkkaDotModule/blob/master/TestAkkaDotModule/TestActors/EventBusTest.cs

```
using Akka.Actor;
using Akka.Event;
using Akka.TestKit;
using AkkaNetCoreTest;
using System;
using Xunit;
using Xunit.Abstractions;

namespace TestAkkaDotModule.TestActors
{
    // https://getakka.net/articles/utilities/event-bus.html

    public class TypeA
    {
        public string Text { get; set; }

    }
    public class TypeB
    {
        public string Text { get; set; }
    }

    public class ActorA : ReceiveActor
    {
        private readonly ILoggingAdapter logger = Context.GetLogger();

        protected IActorRef probe;

        public ActorA(IActorRef _probe)
        {
            probe = _probe;
```

```
        ReceiveAsync<TypeA>(async message =>
        {
            logger.Debug($"InMessage:{message.Text}");

            probe.Tell(message);

        });
    }
}

public class ActorB : ReceiveActor
{
    private readonly ILoggingAdapter logger = Context.GetLogger();

    protected IActorRef probe;

    protected IActorRef reply;

    public ActorB(IActorRef _probe)
    {
        probe = _probe;

        ReceiveAsync<TypeA>(async message =>
        {
            logger.Debug($"InMessage:{message.Text}");

            if(probe!=null) probe.Tell(message);

            if(reply!=null) reply.Tell(message);

        });

        ReceiveAsync<IActorRef>(async actorRef =>
        {
            reply = actorRef;
        });

    }
}

public class ActorC : ReceiveActor
{
    private readonly ILoggingAdapter logger = Context.GetLogger();

    protected IActorRef probe;

    public ActorC(IActorRef _probe)
    {
        probe = _probe;

        ReceiveAsync<TypeA>(async message =>
        {
            logger.Debug($"InMessage:{message.Text}");

            probe.Tell(message);

        });

        ReceiveAsync<TypeB>(async message =>
        {
            logger.Debug($"InMessage:{message.Text}");

            probe.Tell(message);

        });
    }
}

public class EventBusTest : TestKitXunit
{
    protected TestProbe probeA;
```

```
        protected TestProbe probeB;

        protected TestProbe probeC;

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

        public void Setup()
        {
            probeA = this.CreateTestProbe("probeA");
            probeB = this.CreateTestProbe("probeB");
            probeC = this.CreateTestProbe("probeC");

        }

        [Theory(DisplayName = " ")]
        [InlineData(1)]
        public void Test1(int testCount)
        {
            var actorA = Sys.ActorOf(Props.Create(() => new ActorA(probeA)));
            var actorB = Sys.ActorOf(Props.Create(() => new ActorB(probeB)));
            var actorC = Sys.ActorOf(Props.Create(() => new ActorC(probeC)));

            // Link ActorB to ActorC
            actorB.Tell(actorC);

            //Subscribe
            Sys.EventStream.Subscribe(actorA, typeof(TypeA));
            Sys.EventStream.Subscribe(actorB, typeof(TypeA));
            Sys.EventStream.Subscribe(actorC, typeof(TypeB));

            //Publish
            Sys.EventStream.Publish(new TypeA() { Text = "TypeA_Message" } );
            Sys.EventStream.Publish(new TypeB() { Text = "TypeB_Message" });

            //UnitTest
            probeA.ExpectMsg<TypeA>(TimeSpan.FromSeconds(1));
            probeB.ExpectMsg<TypeA>(TimeSpan.FromSeconds(1));
            probeC.ExpectMsg<TypeB>(TimeSpan.FromSeconds(1));
            probeC.ExpectMsg<TypeA>(TimeSpan.FromSeconds(1));
        }
    }
}
```

:

- https://getakka.net/articles/utilities/event-bus.html -
- https://jhleed.tistory.com/163 -