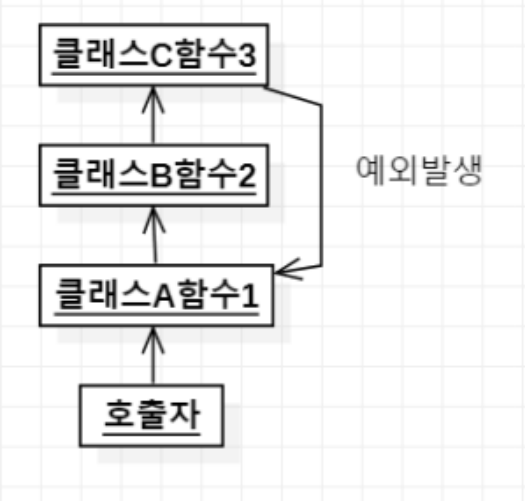
Fault Tolerance

.

? (SupervisorStrategy)

Link :



.

.

, .

B , A . -

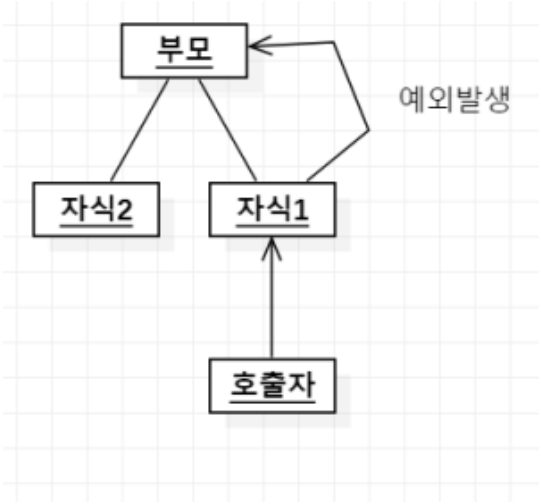
, .

> . (Try Catch .)
>
> . A
>
> .

, .

, . (,)

Actor



, .

,

.

3 (Resume) , (Restart), (Stop)

Supervisor() .

.

- **OneForOne** : . / , . .
- **AllForOne** : . / .

```
using System;
using Akka.Actor;
using Akka.Event;

namespace AkkaNetCore.Actors.Study
{
    public class Child : ReceiveActor
    {
        private int state = 0;

        private readonly ILoggingAdapter logger = Context.GetLogger();

        public Child()
        {
            ReceiveAsync<object>(async message =>
            {
                switch (message)
                {
                    case Exception ex:
                        throw ex;
                    case int x:
                        state = x;
                        break;
                    case "get":
                        Sender.Tell(state);
                        break;
                }
            });
        }
    }

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

        public Supervisor()
        {
            ReceiveAsync<Props>(async p =>
            {
                var child = Context.ActorOf(p); // create child
                Sender.Tell(child); // send back reference to child actor
            });
        }

        protected override SupervisorStrategy SupervisorStrategy()
        {
            return new OneForOneStrategy(
                maxNrOfRetries: 10,
                withinTimeRange: TimeSpan.FromMinutes(1),
                localOnlyDecider: ex =>
                {
                    switch (ex)
                    {
                        case ArithmeticException ae:
                            return Directive.Resume;
                        case NullReferenceException nre:
                            return Directive.Restart;
                        case ArgumentException are:
                            return Directive.Stop;
                        default:
                            return Directive.Escalate;
                    }
                });
        }
    }
}
```

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

namespace AkkaNetCoreTest.Actors
{
    public class SupervisorTest : TestKitXunit
    {
        IActorRef supervisor;

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

        public void Setup()
        {
            supervisor = Sys.ActorOf<Supervisor>("supervisor");
        }

        [Fact(DisplayName = "ArithmeticException ,NullReferenceException ,  .")]
        public void Test1()
        {
            supervisor.Tell(Props.Create<Child>());
            var child = ExpectMsg<IActorRef>();

            //
            child.Tell(42); // set state to 42
            child.Tell("get");
            ExpectMsg(42);

            child.Tell(new ArithmeticException());      // Directive.Resume
            child.Tell("get");
            ExpectMsg(42);

            // ,Child  0.
            child.Tell(new NullReferenceException());   //Directive.Restart
            child.Tell("get");
            ExpectMsg(0);

            //Watch     .( )
            Watch(child);
            child.Tell(new ArgumentException());        //Directive.Stop
            var message1 = ExpectMsg<Terminated>();
            Assert.Equal(message1.ActorRef,child);

            supervisor.Tell(Props.Create<Child>()); // create new child
            var child2 = ExpectMsg<IActorRef>();
            Watch(child2);
            child2.Tell("get"); // verify it is alive
            ExpectMsg(0);

            child2.Tell(new Exception("CRASH"));
            var message2 = ExpectMsg<Terminated>();
            Assert.Equal(message2.ActorRef, child2);
            Assert.Equal(true, message2.ExistenceConfirmed);
        }
    }
}
```