Implementing Persistent Actors

Jason Roberts
.NET MVP

@robertsjason dontcodetired.com

Overview

Understanding persistent actors

Different base classes/receive methods

Install Akka.Persistence.SqlServer

Refactor PlayerActor to use persistent base class

Persisting and recovering messages

In-memory journal

Configuring SQL Server journal store

Player health restored on app restart

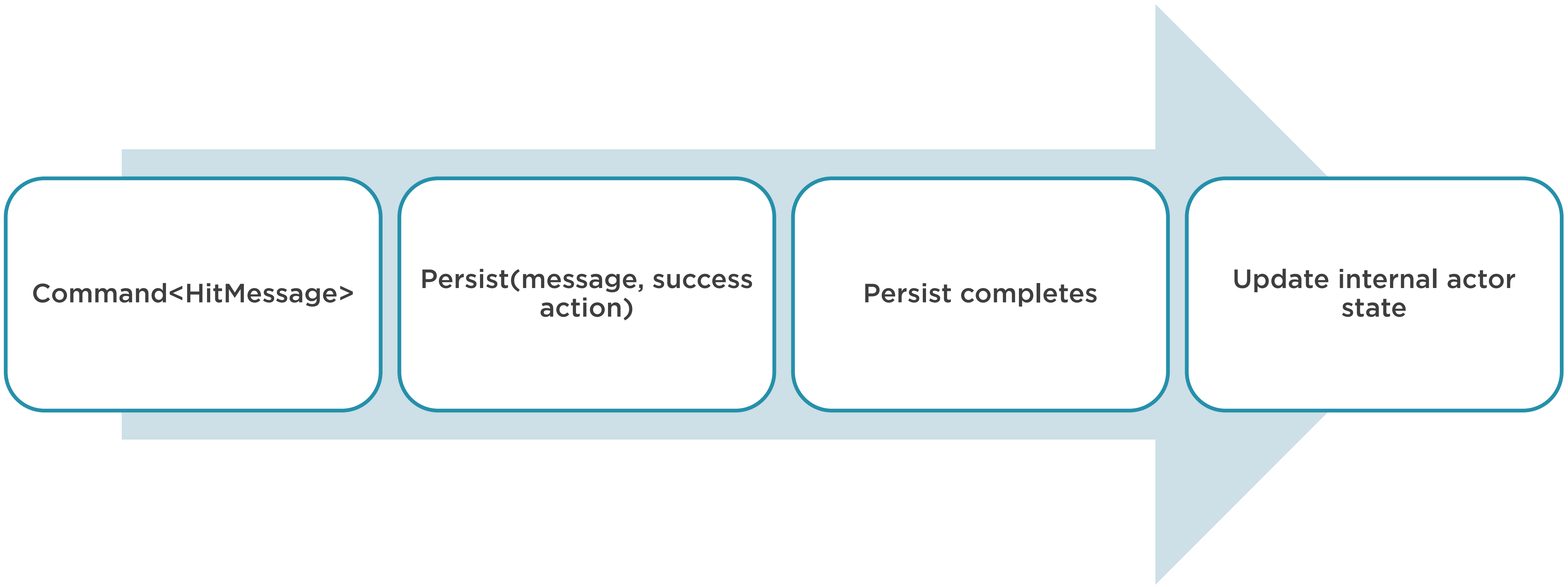
Restoring Players on application restart

Additional properties/methods & considerations

Understanding Persistent Actors

Non-persistent	Persistent
ReceiveActor base class	ReceivePersistentActor base class
Loses internal state on actor restart	Restores internal state on restart
Receive<HitMessage>(...)	Command<HitCommand>(...)
	PersistenceId { get; }
	Persist(...)
	Recover<HitCommand>(...)
	SaveSnapshot(...)
	Recover<SnapshotOffer>(...)

Understanding Persistent Actors



Demo

Getting Started

Install Akka.Persistence.SqlServer NuGet

Install dependencies

E.g. Akka.Persistence

Refactor PlayerActor:

- Inherit ReceivePersistentActor
- Override PersistenceId
- Command<HitMessage>
- Command<DisplayStatusMessage>
- Command<CauseErrorMessage>

Run

State still lost

Demo

Persisting And Recovering Messages

Persist(message, hitMessage => ...)

Recover<HitMessage>(message => ..)

Run console application

Create player

Reduce health

Cause exception

PlayerActor restarts

Health restored

Restart console application

Health state lost

Demo

Configuring SQL Server Journal Store

Empty SQL Server database “PSAkka”

HOCON configuration

Connection string

Initialize journal table automatically

Run console application

Create player

Hit player multiple times

See rows in SQL Server journal table

Cause error, messages replayed from SQL

Restart console application

Create player, messages replayed

Demo

Restoring Players on Application Restart

Refactor PlayerCoordinatorActor :

- Inherit ReceivePersistentActor
- Override PersistenceId
- Command<CreatePlayerMessage>
- Persist(message, createPlayerMessage)
- Recover<CreatePlayerMessage>
- Re-create PlayerActor

Run console application

Add players

Restart console application

Players automatically re-added into game

Additional Properties and Methods

Override OnPersistFailure

- Called when a persist fails
- Actor stopped after method executes
- Default logging of error

Override OnReplaySuccess

- Called after every message replay success

IsRecovering property

- True if actor is recovering events

DeleteMessages(…)

- Removed events from journal
- E.g. after creating snapshot
- Audit/business requirements to keep?

Additional Considerations

Sender property

- Can’t use sender property of message being replayed
- Assumes original sender actor no longer exists
- Sent to dead letters

Safe persistent actor shutdown

- Don’t send PoisonPill
- Some events may not be persisted
- Handle custom message
- Context.Stop(Self);

Summary

Install Akka.Persistence.SqlServer

Refactor PlayerActor to use persistent base class

- Inherit ReceivePersistentActor
- Override PersistenceId
- Command<HitMessage>

Persist(...) & Recover<HitMessage>

Configuring SQL Server journal store

Player health restored on app restart

Refactored PlayerCoordinatorActor to restore players on application restart

Additional properties/methods & considerations

Next:

Thinking in Events and Commands