Next important stepts

1. Verificar que el que detecta cambios en el readmodel (el cliente que hace pool para ver si se hizo su trabajo) tenga un query sencillo en ado, o si no, directamnte en entity framework sea con ChangeDetectionEnabled = false.
2. Hacer que el ReadModelGenerator se encargue de loguear los mensajes que le lleguen.
3. Verificar por que no agregó un elemento en el messageLog. Preguntarse por qué ocurre esto que no se agrega al log: Respuesta. El Log no agrega eventos de readModeling, solamente de proceso interno y recibidos de afuera. Y no hace falta por que los procesos del read model son solo para el read System, no para subscriptores externos
4. Crear en el events del tipo de reporte, o integration, para que el hacer read model sea rapidísimo (puede ser una interfaz, como marcador, esto haría que el read model sea mucho más rápido al no tener que procesar todos los eventos.)
5. Terminar de arreglar el message log, usando solamente los mensajes del origin external.
6. Arreglar que no pueda arrancar rebuilders al mismo tiempo usando el coordinator.
7. Arreglar el event store rebuilder.
8. Limitar la cantidad de mensajes que procesa al mismo tiempo (hacer tests de performance)
9. Testear en la madrugada si los commandos se procesan en IIS express al ser programados
10. LIMPIAR EL COMMAND Y EVENT BUS, esto sirve para procesar los eventos con fecha pendientes. De esta forma se preserva el flujo de trabajos programados al momento de la reconstrucción. Que limpie el command bus y el event bus y lo ponga todo en 0 los generadores de id en la base de datos.
11. Hacer formatos empaquetados para colocar configuraciones con base de datos separados de Log, Engine. ReadModel. La web que hace los queryies debe tener pooling, mientras que el worker no.
12. Crear los database inititializers
13. Hacer que la interfaz web del bus consuslte el estado, si esta haciendo rebuild o no y no muestre opciones de rebuild si esta constuyendo.
14. Arreglar el coordinator, no se ve muy bien.

Next steps

1. Capturar la exception que salta cuando colocamos un comando repetido en el bus. Hasta el momento aparece un DbUpdate Exception, pero lo podemos mejorar si hacemos que al saltar una excpcion se ejecute una consulta en el message log preguntando si existe un command con el mismo id. Si existe el mismo id entonces lanzar una exception customizada de duplicate message. Esto evitaría hacer que un DbUpdate exception que se deba a otro motivo no se oscurezca.

Conventions

Puntos de entrada para implementar un dominio con el Journey Framework:

1. Insertar el domain container en esta sección:
2. 

Arquitectura de un dominio. Assemblies = Layers

1. Domain: Aquí se encuentran los handlers y los aggregates
2. Domain.Commands: Aquí se encuentran los commands. Es el único lugar donde debería conectarse con la capa de aplicación.

Reglas:

1. Todo comando debe generar un evento. Asi se evita el reprocesamiento.
2. Hay que poner un Lock en el handler cuando sea un evento o un comando que va a tener multiples usuarios concurrentes compitiendo por el mismo recurso. El handler debe manejar esto para evitar concurrencias.
3. Cada handler subscriptor, sea un read model generator o un complex event procesor debe ser idempotente, o sea, aunque repetidamente se le pase el mismo evento debe poder ser capaz de procesarlo una sola vez.
4. Las materializaciones deben ser sencillas. Deben ser eventos que solamente necesitan ser copiados en una tabla, sin procesarlos.
5. Para los comandos de los Sagas, debe usar el mismo correlation Id de un evento, o el mismo id del comando. Solamente asi se evitarían las duplicaciones al reconstruir el EventStore.
6. Todos los que se subscriban en al bus (handlers) deben asegurarse de primero escribir en el message log (buscar si existe su registro, y si no, grabar y hacer lo que tienen que hacer).
7. Entity Framework Query Optimization

El disable de entity framework para que sea más rápido.

public SimpleInventarioDao(Func<SimpleInventarioDbContext> contextFactory)

        {

            // performance optimizer.

            this.contextFactory = () =>

            {

                var context = contextFactory.Invoke();

                context.Configuration.AutoDetectChangesEnabled = false;

                return context;

            };

        }

Machine generated alternative text:
PROJECT 
ocøuG 
TEAM 
TEST 
ANALYZE 
public MerpContext 
public Database() 
this . Context 
Context get; private set; 
new Merpcontext(); 
this .Context. Configuration.AutoDetectChangesEnab1ed false; 
*region IDatabase Members 
public IQueryablecActivityType) ActivityTypes 
get return Context. ActivityTypes; 
public IQueryabIe«BusinesslJnit) BusinesslJnits 
get return Context. BusinessUnits; ) 
31:31 / 

Message Receiver.

Ahora se pueden ver como dos transcciones de vida larga en la base de datos a los procesos que miran las tablas del bus:



SOBRE PORQUE NO USO BUS EN MEMORIA – 3 RAZONES

1. Se pierde la capacidad de programar tareas (poniendo fecha de procesamiento del mensaje en el bus).
2. Se pierde la capacidad de ejecutar largos procesos de análisis de eventos. Si fuera en memoria se tendría que ejecutar todo primero antes de mostrarle cosas al usuario. Esto quizás si no fuera muy largo sería rapidísimo.
3. Es más difícil el multiusuario (cada vez que se hace commit ya puede un usuario ganar un turno en el event store)

Testing - EN EL CONNECTION STRING DEL BUS EL POOLING DEBE SER FALSE: Funciona

TEST: Soporta 1000 usuarios en simultaneo sobre el mismo aggregate!