EF Core Interceptors

Кулага Иван
Senior .NET Developer

Agenda

- What is it?
- How we could use it?
- What’s new in EF Core 7 Interceptors
- Demo
- Q&A

Use case: audit

```
class Order
{
    public int Id { get; set; }
    public string Name { get; set; }

    public DateTime CreatedAt { get; set; }
    public string CreatedBy { get; set; }
}
```

Use case: audit

```
public Order CreateOrder(string name, string userId)
{
    return new Order
    {
        Name = name,
        CreatedAt = DateTime.UtcNow,
        CreatedBy = userId,
    }
}
```

Use case: audit

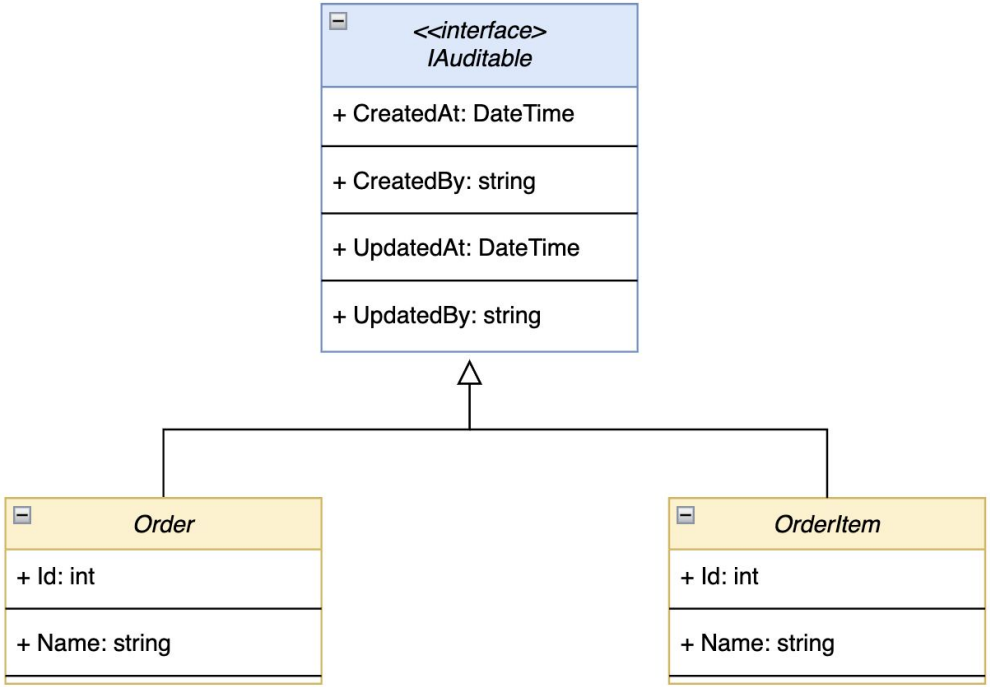
```
public class OrderItem
{
    public int Id { get; set; }
    public string Name { get; set; }
    public int OrderId { get; set; }

    public DateTime CreatedAt { get; set; }
    public string CreatedBy { get; set; }
    public DateTime UpdatedAt { get; set; }
    public string UpdatedBy { get; set; }
}
```

Use case: audit

```
public OrderItem CreateOrderItem(string name, string userId)
{
    return new OrderItem
    {
        Name = name,
        CreatedAt = DateTime.UtcNow,
        CreatedBy = userId,
        UpdatedAt = DateTime.UtcNow,
        UpdatedBy = userId
    }
}
```

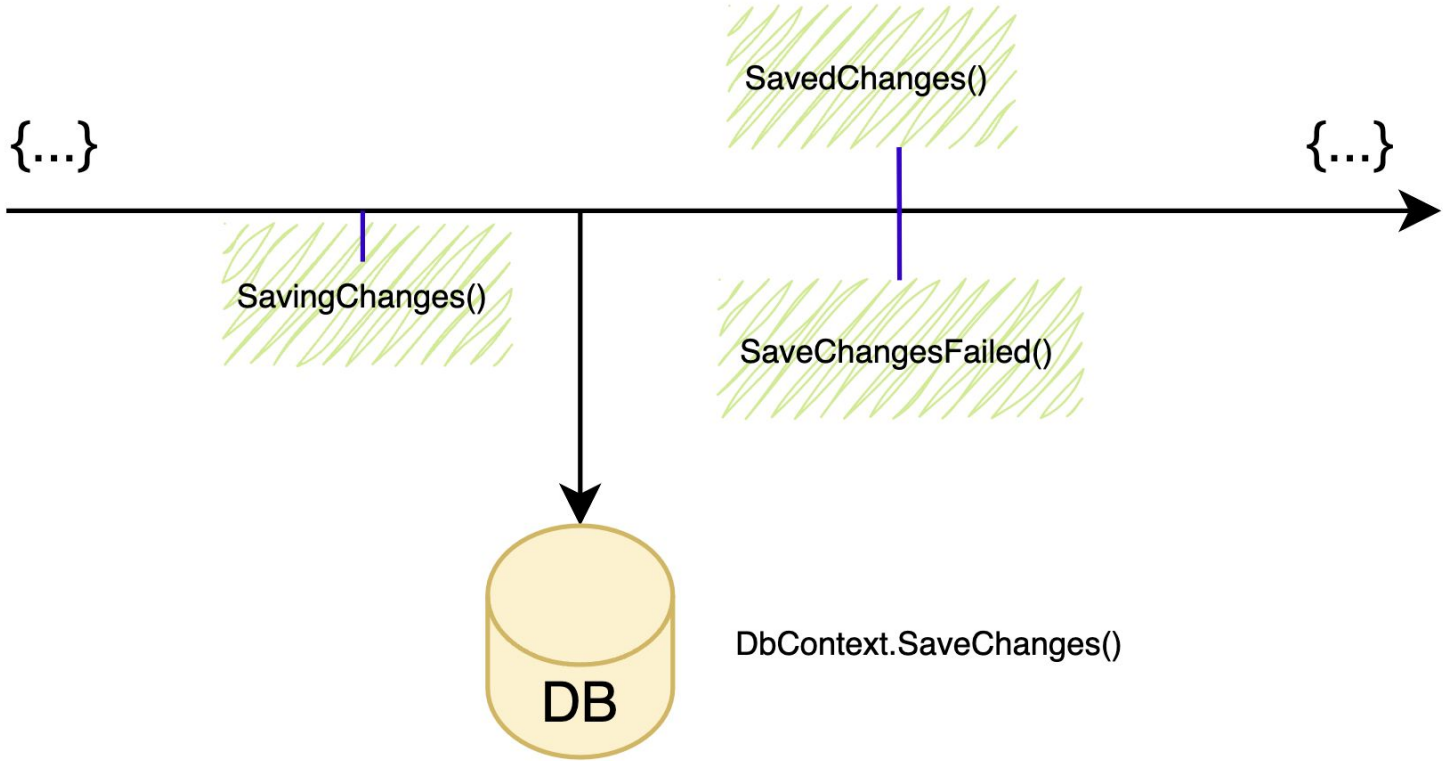
Use case: audit



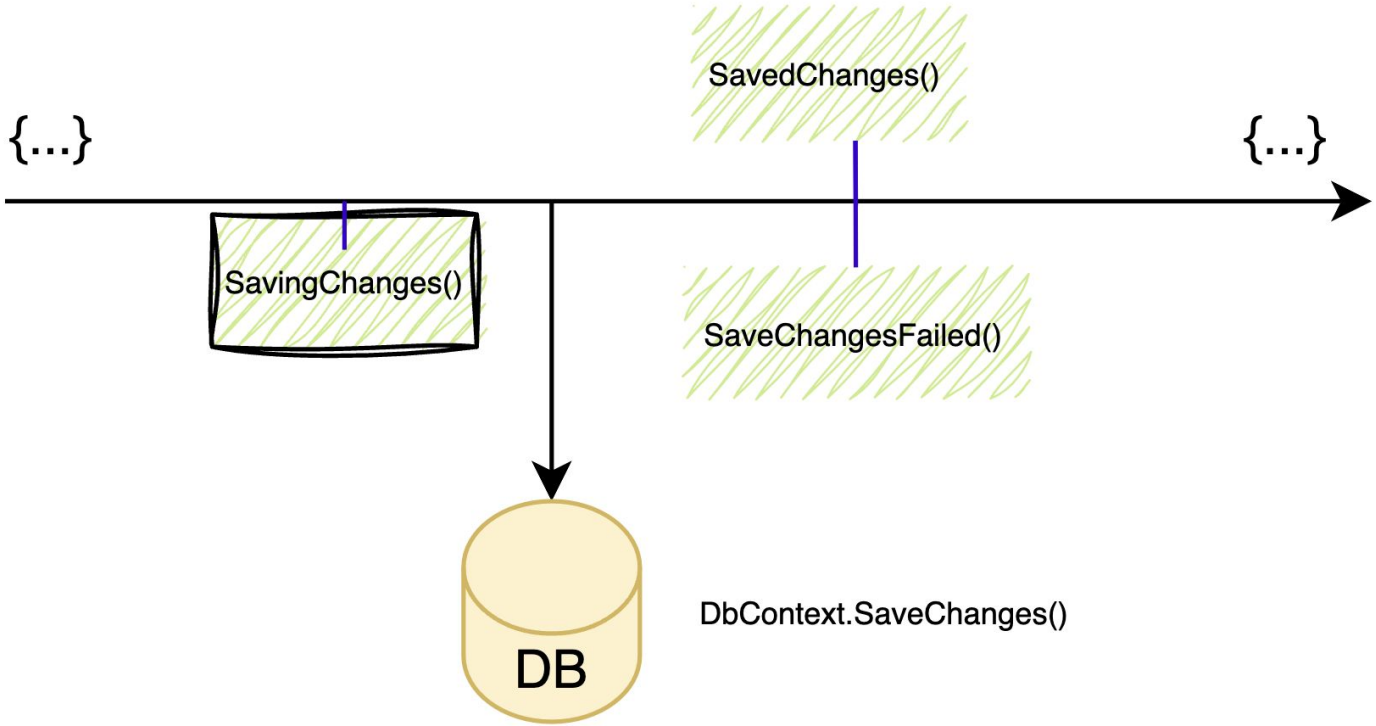
What is it?

Interceptors enable interception, modification, and/or suppression of EF Core operations. This includes low-level database operations such as executing a command, as well as higher-level operations, such as calls to SaveChanges.

ISaveChangesInterceptor



ISaveChangesInterceptor



Use case: audit

```
public override InterceptionResult<int> SavingChanges(eventData, ...)
{
    var entries = eventData.Context.ChangeTracker
        .Entries()
        .Where(e => e.Entity is IAuditable)
        .ToList();

    foreach (var entry in entries)
    {
        var now = DateTime.UtcNow;
        var auditable = entry.Entity as IAuditable;

        if (entry.State == EntityState.Modified)
        {
            auditable.UpdatedBy = UserId;
            auditable.UpdatedAt = now;
        }

        if (entry.State == EntityState.Added)
        {
            auditable.CreatedBy = UserId;
            auditable.CreatedDate = now;
            auditable.UpdatedBy = UserId;
            auditable.UpdatedAt = now;
        }
    }
    return base.SavingChanges(eventData, result);
}
```

Interceptor types

- Command
- Query
- DbConnection
- SaveChanges

Command/Query/Connection interceptor

These interceptors are used to intercept and manipulate commands that are sent to the database. They can be used for

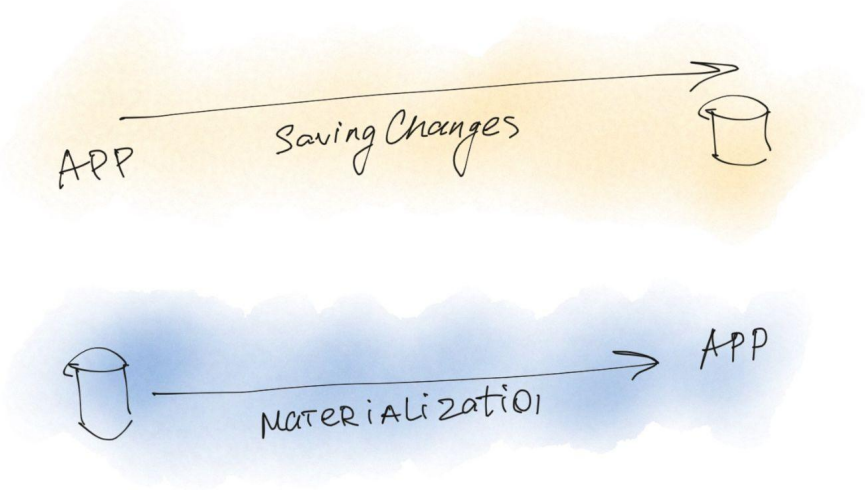
- logging
- auditing
- modifying the command/connection before it is executed.

Advanced usage

- Data validation
- Caching

EF Core 7: what's new?

IMaterializationInterceptor (EF6 port)

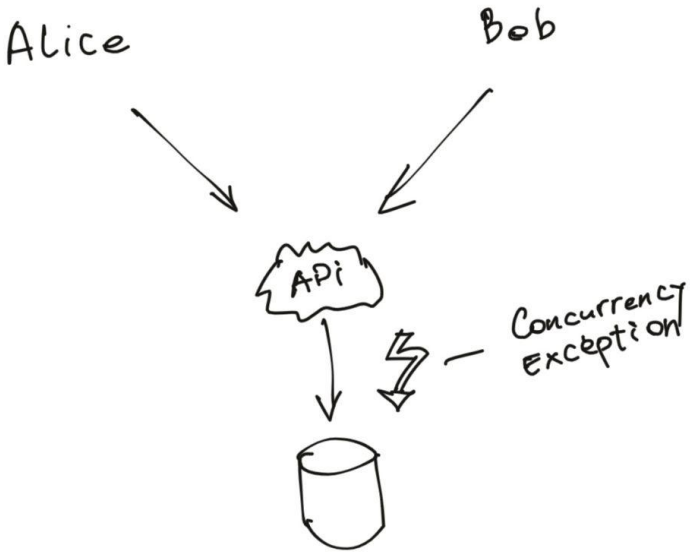


IMaterializationInterceptor usage

- Entity mutation before usage (converters alternative, Encryption, Optimization)
- Logging

ISaveChangesInterceptor.ThrowingConcurrencyException

- Called when concurrency exception is thrown



ISaveChangesInterceptor.ThrowingConcurrencyException

```
public InterceptionResult ThrowingConcurrencyException(
    ConcurrencyExceptionEventData eventData,
    InterceptionResult result)
{
    if (eventData.Entries.All(e => e.State == EntityState.Deleted))
    {
        Console.WriteLine("Suppressing Concurrency violation for command:");
        Console.WriteLine(((RelationalConcurrencyExceptionEventData) eventData).

        return InterceptionResult.Suppress();
    }

    return result;
}
```

LINQ expression tree interception

```
SELECT "c"."Id", "c"."City", "c"."Name", "c"."PhoneNumber"
FROM "Customers" AS "c"
```

INTO

```
SELECT "c"."Id", "c"."City", "c"."Name", "c"."PhoneNumber"
FROM "Customers" AS "c"
WHERE "c"."Name" = 'Ivan'
```

LINQ expression tree interception

WARNING: hard to maintain

```
public class KeyOrderingExpressionInterceptor : IQueryExpressionInterceptor
{
    public Expression ProcessingQuery(Expression queryExpression, QueryExpressionEve
        => new KeyOrderingExpressionVisitor().Visit(queryExpression);

    private class KeyOrderingExpressionVisitor : ExpressionVisitor
    {
        private static readonly MethodInfo ThenByMethod
            = typeof(Queryable).GetMethods()
                .Single(m => m.Name == nameof(Queryable.ThenBy) && m.GetParameters()

        protected override Expression VisitMethodCall(MethodCallExpression? methodCa
        {
            var methodInfo = methodCallExpression!.Method;
            if (methodInfo.DeclaringType == typeof(Queryable)
                && methodInfo.Name == nameof(Queryable.OrderBy)
                && methodInfo.GetParameters().Length == 2)
            {
                var sourceType = methodCallExpression.Type.GetGenericArguments()[0];
                if (typeof(IHasIntKey).IsAssignableFrom(sourceType))
                {
                    var lambdaExpression = (LambdaExpression)((UnaryExpression)metho
                    var entityParameterExpression = lambdaExpression.Parameters[0];

                    return Expression.Call(
                        ThenByMethod.MakeGenericMethod(
                            sourceType,
                            typeof(int)),
                        methodCallExpression,
                        Expression.Lambda(
                            typeof(Func<,>).MakeGenericType(entityParameterExpressio
                            Expression.Property(entityParameterExpression, nameof(IH
                            entityParameterExpression));
                }
            }

            return base.VisitMethodCall(methodCallExpression);
        }
    }
}
```

Other changes

- Interception for connections *before* checking if the connection string has been set
- Interception for when EF Core has finished consuming a result set, but before that result set is closed
- Interception for creation of a `DbConnection` by EF Core
- Interception for `DbCommand` after it has been initialized

Demo

Best practices

- Changes that are common and related to infrastructure
- Avoid using interceptors for business logic
- Keep the interceptors lightweight
- Use them only when necessary.
- Avoid using interceptors to modify the structure of the database (Duh)

Summary

Interceptors are a powerful feature of EF Core that can be used

- to add additional functionality to an application without modifying the application's code
- for logging
- for auditing
- for security
- for performance optimization.

EF 7 Interceptors

- Materialization (data flow in and out)
- Concurrency handling
- Query interception
- Different lifecycle for DbConnection/DbCommand

Resources

- EF Core documentation on Interceptors: **https://docs.microsoft.com/en-us/ef/core/querying/interceptors**
- EF Core Interceptors sample: **https://github.com/dotnet/efcore/tree/main/samples/Interception**
- **https://devblogs.microsoft.com/dotnet/announcing-ef7-preview7-entity-framework/**

Q&A

Please feel free to ask any questions about the topic.