

## 第31期-多租户数据隔离与软删除机制

2021年5月8日 22:16

### 租户信息异步线程上下文

CurrentTenant.cs -> CurrentTenantAccessor.cs -> AsyncLocal<TenantInfo?>

### 使用中间件在提取租户信息

TenantMiddleware 拦截请求并使用 CurrentTenant 设置当前工作线程的的租户信息。

```
app.UseAuthentication();  
app.UseTenantMiddleware();
```

租户中间件一定要配置在身份认证之后，因为只有身份认证后令牌中的租户ID才会绑定到用户的声明信息中，才能读取到，才能被租户中间件提取声明信息。

### 多租户数据隔离筛选器

```
modelBuilder.Entity(entityType.ClrType).AddQueryFilter<IMultiTenant>(e =>  
e.TenantId == currentTenant.Id);
```

### 为实体自动设置租户编号

```
private static void MultiTenancyTracking(DbContext dbContext)  
{  
    var tenantedEntries = dbContext.ChangeTracker.Entries<IMultiTenant>  
().Where(entry => entry.State == EntityState.Added);  
    var currentTenant = dbContext.GetService<ICurrentTenant>();  
    tenantedEntries?.ToList().ForEach(entityEntry =>  
    {  
        entityEntry.Entity.TenantId ??= currentTenant.Id;  
    });  
}
```

### 软删除全局筛选器

```
modelBuilder.Entity(entityType.ClrType).AddQueryFilter<ISoftDelete>(e => !  
e.IsDeleted);
```

## 软删除状态跟踪器

```
private static void SoftDeleteTracking(DbContext dbContext)
{
    var deletedEntries = dbContext.ChangeTracker.Entries().Where(entry =>
entry.State == EntityState.Deleted && entry.Entity is ISoftDelete);
    deletedEntries?.ToList().ForEach(entityEntry =>
    {
        entityEntry.Reload();
        entityEntry.State = EntityState.Modified;
        ((ISoftDelete)entityEntry.Entity).IsDeleted = true;
    });
}
```

## 保存实体变更时分发领域事件

```
private async Task DispatchDomainEventsAsync(DbContext dbContext, CancellationToken
cancellation_token = default)
{
    var domainEntities = dbContext.ChangeTracker.Entries<IDomainEvents>().Select(e
=> e.Entity);
    var domainEvents = domainEntities.SelectMany(x => x.DomainEvents).ToList();
    domainEntities.ToList().ForEach(entity => entity.ClearDomainEvents());
    foreach (var domainEvent in domainEvents)
    {
        await _mediator.Publish(domainEvent, cancellation_token);
    }
}
```

## 使用保存变更拦截器横切关注点

```
public class CustomSaveChangesInterceptor : SaveChangesInterceptor
```

```
IMediator mediator = serviceProvider.GetService<IMediator>() ?? new NullMediator();
optionsBuilder.AddInterceptors(new CustomSaveChangesInterceptor(mediator));
```

## 生成基于多租户和软删除的产品演示数据

```
public class Product : BaseAggregateRoot<Guid>, ISoftDelete, IMultiTenant
```

```
public async Task SeedAsync(IServiceProvider serviceProvider)
```

```

{
    if (await _productRepository.GetCountAsync() <= 0)
    {
        for (int i = 1; i < 30; i++)
        {
            Guid? tenantId = null;

            if (i >= 10 && i < 20)
            {
                tenantId = Guid.Parse($"f30e402b-9de2-4b48-9ff0-c073cf499102");
            }

            if (i >= 20 && i < 30)
            {
                tenantId = Guid.Parse($"f30e402b-9de2-4b48-9ff0-c073cf499103");
            }

            var product = new Product { Name = $"Product{i.ToString().PadLeft(2,
'0')}}", TenantId = tenantId };
            await _productRepository.InsertAsync(product, true);
        }
    }
}

```