

# 第37期-仓储模式与扩展

2020年9月27日 9:14

## 自定义仓储

```
public interface IDogRepository : IRepository<Dog, int>
{
    Task DeleteDogsByAge(int age);
}

public class EfCoreDogRepository : EfCoreRepository<DogStoreDbContext, Dog, int>,
IDogRepository
{
    public EfCoreDogRepository(IDbContextProvider<DogStoreDbContext>
dbContextProvider) : base(dbContextProvider)
    {
    }

    public async Task DeleteDogsByAge(int age)
    {
        await DbContext.Database.ExecuteSqlRawAsync($"DELETE FROM AbpDogs
WHERE Age = {age}");
    }
}
```

现在，可注入 IDogRepository 直接使用了，并使用 DeleteDogsByAge 方法。

## 配置自定义仓储

即使创建了自定义仓储，默认通用仓储仍然可用，但内部继续使用 EfCoreRepository 的实现，要将自定义实现用于默认通用仓储，进行如下配置。

```
context.Services.AddAbpDbContext<MyDbContext>(options =>
{
    options.AddDefaultRepositories();

    //Replaces IRepository<Dog, int>
    options.AddRepository<Dog, EfCoreDogRepository>();
});
```

这样的好处是能够让默认通用仓储使用 override 覆盖 EfCoreRepository 的实现。

```
public override Task<Dog> InsertAsync(Dog entity, bool autoSave = false,
Cancellation token cancellationToken = default)
{
}
```

```

        return base.InsertAsync(entity, autoSave, cancellationTokens);
    }

```

## 配置扩展属性

```

ObjectExtensionManager.Instance.MapEfCoreProperty<Dog, string>("Remark",
    (entityBuilder, propertyBuilder) =>
    { propertyBuilder.HasMaxLength(60); });

```

```

builder.Entity<YourEntity>(b =>
{
    b.ConfigureObjectExtensions();
    //...
});

```

如果您已调用 `ConfigureByConvention()` 扩展方法，则 ABP Framework 在内部会自动调用 `ConfigureObjectExtensions` 方法。

## 设置默认仓储实现类

默认的通用仓储的实现是 `EfCoreRepository` 类，你可以创建自己的实现，并将其做为默认实现。

```

public class MyRepositoryBase<TEntity> : EfCoreRepository<DogStoreDbContext,
TEntity> where TEntity : class, IEntity
{
    public MyRepositoryBase(IDbContextProvider<DogStoreDbContext>
dbContextProvider) : base(dbContextProvider)
    {
    }
}

```

```

public class MyRepositoryBase<TEntity, TKey> : EfCoreRepository<DogStoreDbContext,
TEntity, TKey> where TEntity : class, IEntity<TKey>
{
    public MyRepositoryBase(IDbContextProvider<DogStoreDbContext>
dbContextProvider) : base(dbContextProvider)
    {
    }
}

```

```

context.Services.AddAbpDbContext<BookStoreDbContext>(options =>
{
    options.SetDefaultRepositoryClasses(
        typeof(MyRepositoryBase<, >),
        typeof(MyRepositoryBase<>));
});

```

## 为默认仓储设置 DbContext 类或接口

```
public interface IDogStoreDbContext
{
    DbSet<Dog> Dogs { get; set; }
}
```

IDogStoreDbContext接口是由DogStoreDbContext实现的，然后你可以使用AddDefaultRepositories的泛型重载。

```
context.Services.AddAbpDbContext<DogStoreDbContext>(options =>
{
    options.AddDefaultRepositories<IDogStoreDbContext>();
});
```

现在，你的自定义仓储也可以使用 IDogStoreDbContext接口：

```
public class EfDogRepository : EfCoreRepository<IDogStoreDbContext, Book, Guid>,
IDogRepository
{
    //...
}
```

**使用DbContext接口的一个优点是它可以被其他实现替换。**

## 替换默认仓储 DbContext 类或接口

```
context.Services.AddAbpDbContext<OtherDbContext>(options =>
{
    options.ReplaceDbContext<IDogStoreDbContext>();
});
```

## 异步操作

### 用法1：在EFCore上异步查询

```
var dog = _dogRepository.Where(d => d.Name=="d1").ToListAsync();
```

### 用法2：MongoDB异步查询

```
var dog= ((IMongoQueryable<Dog>)_dogRepository .Where(p =>"d1")).ToListAsync();
```

### 用法3：使用 IAsyncQueryableExecuter 接口

```
var query = _dogRepository.Where(p => p.Name.Contains(name)).OrderBy(p => p.Name);  
List<Dog> dogs = await _asyncQueryableExecuter.ToListAsync(query);
```

注意：ApplicationService 和 DomainService 基类已经预属性注入了 AsyncExecuter 属性，所以你可直接使用。