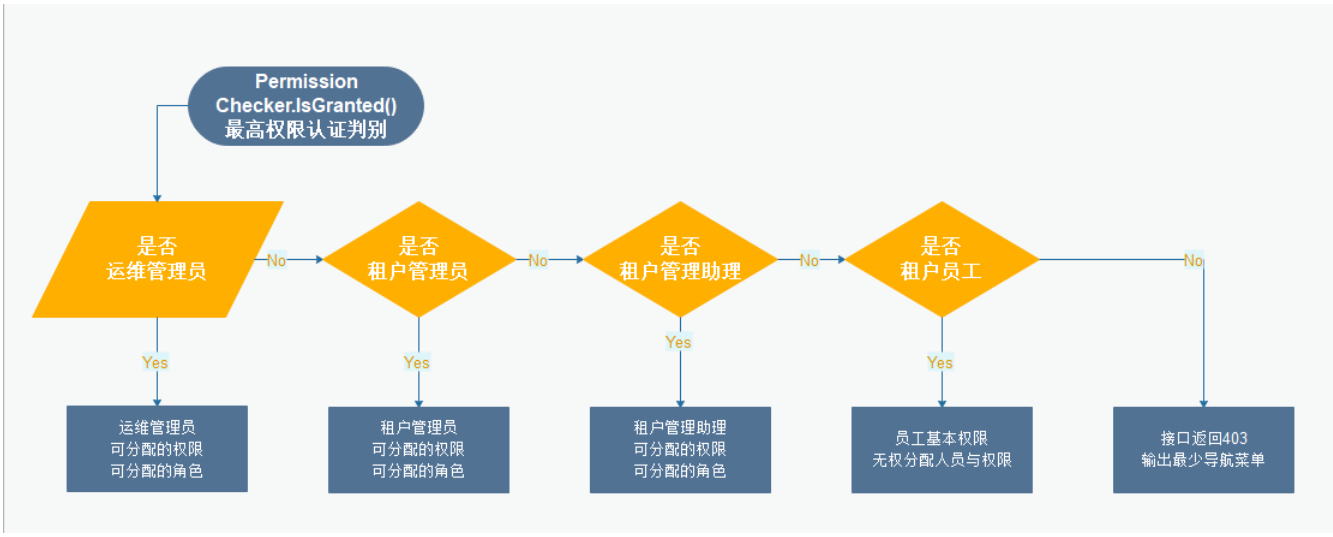
Microsoft.AspNet.Identity;

```
using Microsoft.AspNet.Identity;

private async Task<UserPermissionCacheItem> GetUserPermissionCacheItemAsync(long userId)
```

策略权限==》Bool判定

```
//控制器or类的权限
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method, AllowMultiple = true)]
public class AbpAuthorizeAttribute : Attribute, IAbpAuthorizeAttribute
{
 public virtual async Task AuthorizeAsync(IEnumerable<IAbpAuthorizeAttribute>
authorizeAttributes)
    {
       ......
        foreach (var authorizeAttribute in authorizeAttributes)
        {//【原始方案】遍历所有授权特性
        //通过 IPermissionChecker 来验证用户是否拥有这些特性所标注的权限
            await PermissionChecker.AuthorizeAsync
            (authorizeAttribute.RequireAllPermissions, authorizeAttribute.Permissions);
        }
        //using代码块限定，决定对操作的执行
        using (CurrentUnitOfWork.DisableFilter(AbpDataFilters.MayHaveTenant,
AbpDataFilters.MustHaveTenant))
           {
               var usersInfo = UserManager.Users;
           }
    }

}
```




```
0 个引用|0 异常
protected override IQueryable<User> ApplySorting(IQueryable<User> query, PagedResultRequestDto input)
{
    //SortedDictionary<string, object> DgDict = new SortedDictionary<string, object>();
    bool canAssignRolesFromAdmin = PermissionChecker.IsGranted(
        UserIdentifier.Parse(AbpClaimTypes.UserId), PermissionNames.Pages_Tenants);
    bool canAssignRolesFromRQAdmin = PermissionChecker.IsGranted(
        UserIdentifier.Parse(AbpClaimTypes.UserId), PermissionNames.Pages_Admin);
    bool canAssignRolesFromRQAssitant = PermissionChecker.IsGranted(
        UserIdentifier.Parse(AbpClaimTypes.UserId), PermissionNames.Pages_RQAssitant);
    //List<Role> RolescanAssigned = allRoles;
    int myTopRoleBack = 0;

    if (canAssignRolesFromAdmin)...
    else if (canAssignRolesFromRQAdmin)
    {   //来着租户系统管理员
        myTopRoleBack = 1;
        //DgDict.Add("myTopRoleBack", "canAssignRolesFromRQAdmin");
        return query
            .Where(r=>r.Surname!="admin")
            .Where(r => Convert.ToInt32(r.Surname) >= 1000)
            .OrderBy(r => r.UserName);
    }
    else if (canAssignRolesFromRQAssitant)...
    else
    {
        //DgDict.Add("myTopRoleBack", null);
        return null;
    }
}
```

```
3 个引用|0 项更改|0 名作者，0 项更改
public interface IRepository : ITransientDependency
{

}
```

微信数据包的泛型

```
public WxPayData()...

//采用排序的Dictionary的好处是方便对数据包进行签名，不用再签名之前再做一次排序
private SortedDictionary<string, object> m_values = new SortedDictionary<string, object>();
```

```
* @param WxPayData inputObj 提交给关闭订单API的参数
* @param int timeOut 接口超时时间
* @throws WxPayException
* @return 成功时返回，其他抛异常
*/
0 个引用|0 项更改|0 名作者，0 项更改
public static WxPayData CloseOrder(WxPayData inputObj, int timeOut = 6)
{
    string url = "https://api.mch.weixin.qq.com/pay/closeorder";
    //检测必填参数
    if(!inputObj.IsSet("out_trade_no"))
    {
        throw new WxPayException("关闭订单接口中，out_trade_no必填！");
    }

    inputObj.SetValue("appid",WxPayConfig.GetConfig().GetAppID());//公众账号ID
    inputObj.SetValue("mch_id",WxPayConfig.GetConfig().GetMchID());//商户号
    inputObj.SetValue("nonce_str",GenerateNonceStr());//随机字符串
    inputObj.SetValue("sign_type", WxPayData.SIGN_TYPE_HMAC_SHA256);//签名类型
    inputObj.SetValue("sign", inputObj.MakeSign());//签名
    string xml = inputObj.ToXml();

    var start = DateTime.Now;//请求开始时间

    string response = HttpService.Post(xml, url, false, timeOut);

    var end = DateTime.Now;
    int timeCost = (int)((end - start).TotalMilliseconds);

    WxPayData result = new WxPayData();
    result.FromXml(response);

    ReportCostTime(url, timeCost, result);//测速上报

    return result;
}
```

分页GetAll()

```
public class PagedResultDto<T> : ListResultDto<T>, IPagedResult<T>, IListResult<T>, IHasTotalCount
{
    public PagedResultDto();
    public PagedResultDto(int totalCount, IReadOnlyList<T> items);

    public int TotalCount { get; set; }
}
```

```
            .OrderByDescending(t => t.CreationTime);
if (!string.IsNullOrEmpty(input.Sorting))//排序字段是否有值
    query = query.OrderBy(t => t.Sorting);
else
{
    query = query.OrderByDescending(t => t.CreationTime);
}

var task = query.ToList();
var taskcount = task.Count;  //数据总量

var tasklist = task.Skip((input.PageIndex - 1) * input.PageSize)
    .Take(input.PageSize).ToList();  //获取目标页数据

var result = new PagedResultDto<SearchInspectionDto>
    (taskcount, tasklist.MapTo<List<SearchInspectionDto>>());

return result;
```

两种Mapper

Plan.A AutoMapper

```
public void UpdateMission(DepartmentInfoDto input, int id)
{
    var task = _userRepository.GetAll().Where(t => t.DepartmentID == input.DepartmentID);
    var result = Mapper.Map<DepartmentInfo>(task);
    if (task != null)
    { _userRepository.Update(result);
    }
}
```

Plan.B Dto赋值

> **移除重复内容** 概念：本文中的"移除重复内容"是指把一些很多地方都用到的逻辑提炼出来，然后提供给调用者统一调用。

```
public IList<SelectDto> GetLogisticsState()
{
    var task = (from t in  StateInfoRepository.GetAll()
                select new SelectDto
                {
                    Id = t.Id,
                    Name = t.Meaning

                }).OrderBy(r => r.Id).ToList();
    return task;
}
```

```
public class SelectDto : EntityDto
{
    14 个引用|0 异常
    public new int Id { get; set; }
    11 个引用|0 异常
    public string Name { get; set; }
}
0 个引用
public class SelectLongDto : EntityDto<long>
{
    0 个引用|0 异常
    public new long Id { get; set; }
    0 个引用|0 异常
    public string Name { get; set; }
}
15 个引用
public class SelectStringDto : EntityDto<string>
{
    5 个引用|0 异常
    public new string Id { get; set; }
    5 个引用|0 异常
    public string Name { get; set; }
}
```

```
public IList<SelectStringDto> GetAllMession()
{
    var task = (from t in _T_ExpressTypeRepository.GetAll()
                .Where(t=>t.IsDefaultShow==true)

                select new SelectStringDto
                {
                    Id = t.ExpressNo,
                    Name = t.ExpressName
                }
                ).ToList();
    return task;
}
```

Plan.C 匿名

```
var now = DateTime.UtcNow;

var jwtSecurityToken = new JwtSecurityToken(
    issuer: _configuration.Issuer,
    audience: _configuration.Audience,
    claims: claims,
    notBefore: now,
    expires: now.Add(expiration ?? _configuration.Expiration),
    signingCredentials: _configuration.SigningCredentials
);

return new JwtSecurityTokenHandler().WriteToken(jwtSecurityToken);
```

仓储泛型

泛型约束

怎么解决类型不安全的问题呢？那就是使用泛型约束。 所谓的泛型约束，实际上就是约束的类型T。使T必须遵循一定的规则。比如T必须继承自某个类，或者T必须实现某个接口等等。那么怎么给泛型指定约束？其实也很简单，只需要where关键字，加上约束的条件。 泛型约束总共有五种。

1. 基类约束

   基类约束时，基类不能是密封类，即不能是sealed类。

   sealed类表示该类不能被继承，在这里用作约束就无任何意义，因为sealed类没有子类。

2. 接口约束
3. 引用类型约束 class

   引用类型约束保证T一定是引用类型的。

4. 值类型约束 struct

   值类型约束保证T一定是值类型的。

5. 无参数构造函数约束 new()

   泛型约束也可以同时约束多个

约束	s说明
T：结构	类型参数必须是值类型
T：类	类型参数必须是引用类型；这一点也适用于任何类、接口、委托或数组类型。
T：new()	类型参数必须具有无参数的公共构造函数。 当与其他约束一起使用时，new() 约束必须最后指定。
T：<基类名>	类型参数必须是指定的基类或派生自指定的基类。
T：<接口名称>	类型参数必须是指定的接口或实现指定的接口。 可以指定多个接口约束。 约束接口也可以是泛型的。

泛型仓储-基类/接口

```
public class EfCoreRepositoryBase<TDbContext, TEntity>
    : EfCoreRepositoryBase<TDbContext, TEntity, int>, IRepository<TEntity>

    where TEntity : class, IEntity<int>
    where TDbContext : DbContext
{
    0 个引用|0 项更改|0 名作者，0 项更改|0 异常
    public EfCoreRepositoryBase(IDbContextProvider<TDbContext> dbContextProvider)
        : base(dbContextProvider)
    {
    }
}
```

```
/// <summary>
/// Implements IRepository for Entity Framework.
/// </summary>
/// <typeparam name="TDbContext">DbContext which contains <typeparamref name="TEntity"/>.</typeparam>
/// <typeparam name="TEntity">Type of the Entity for this repository</typeparam>
/// <typeparam name="TPrimaryKey">Primary key of the entity</typeparam>
7 个引用|0 项更改|0 名作者，0 项更改
public class EfCoreRepositoryBase<TDbContext, TEntity, TPrimaryKey> :
    AbpRepositoryBase<TEntity, TPrimaryKey>,
    ISupportsExplicitLoading<TEntity, TPrimaryKey>,
    IRepositoryWithDbContext

    where TEntity : class, IEntity<TPrimaryKey>
    where TDbContext : DbContext
{
    /// <summary>
    /// Gets EF DbContext object.
    /// </summary>
    12 个引用|0 项更改|0 名作者，0 项更改|0 异常
    public virtual TDbContext Context => _dbContextProvider.GetDbContext(MultiTenancySide);
```

```
/// </summary>
/// <typeparam name="TEntity">Type of the Entity for this repository</typeparam>
/// <typeparam name="TPrimaryKey">Primary key of the entity</typeparam>
10 个引用||smail ÇAGDAŞ, 252 天前|1 名作者，1 项更改|1 个工作项
public abstract class AbpRepositoryBase<TEntity, TPrimaryKey>
    : IRepository<TEntity, TPrimaryKey>, IUnitOfWorkManagerAccessor

    where TEntity : class, IEntity<TPrimaryKey>
{
```

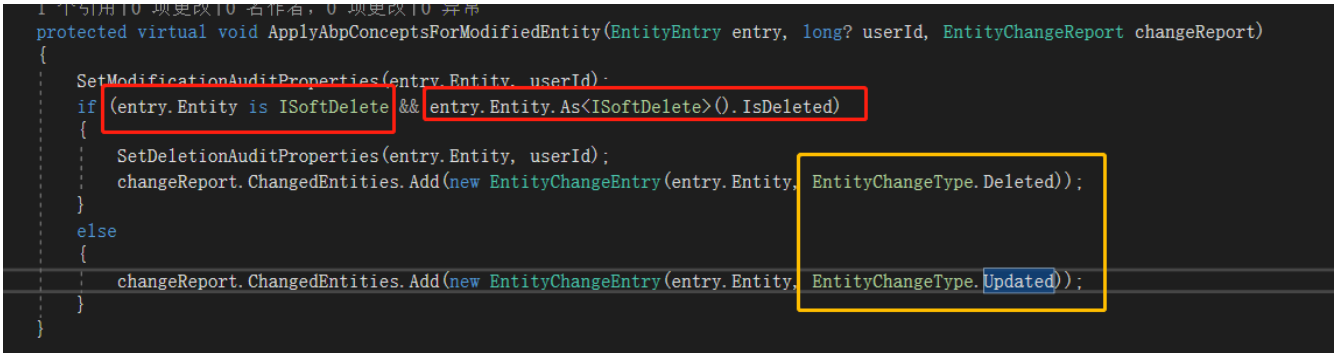
```
99+ 引用|0 项更改|0 名作者，0 项更改
public interface IRepository<TEntity, TPrimaryKey>
    : IRepository where TEntity : class, IEntity<TPrimaryKey>
{
    Select/Get/Query

    Insert

    Update

    Delete

    Aggregates
}
```

软删除

> **分离职责** 概念：本文中的“分离职责”是指当一个类有许多职责时，将部分职责分离到独立的类中，这样也符合面向对象的五大特征之一的单一职责原则，同时也可以使代码的结构更加清晰，维护性更高。
>
> 总结：这个重构经常会用到，它和之前的“移动方法”有几分相似之处，让方法放在合适的类中，并且简化类的职责，同时这也是面向对象五大原则之一和设计模式中的重要思想。

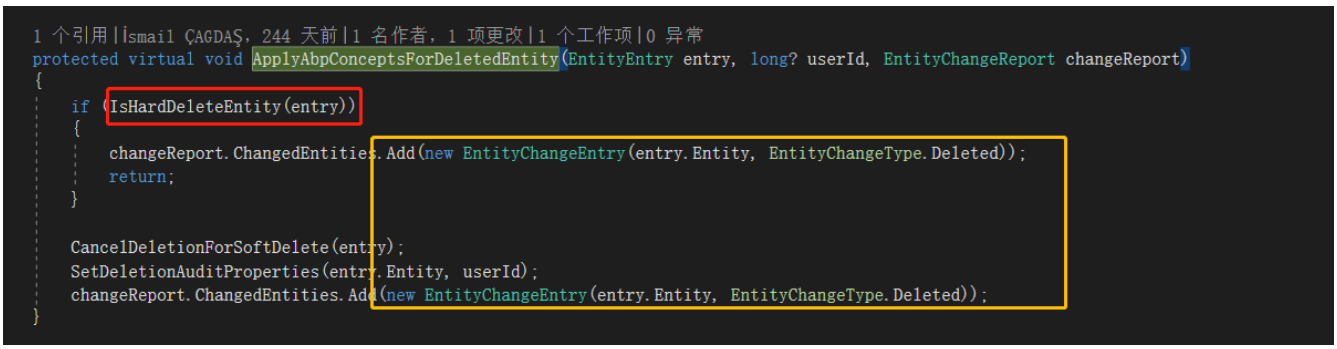
> **提取方法对象** 概念：本文中的“提取方法对象”是指当你发现一个方法中存在过多的局部变量时，你可以通过使用“提取方法对象”重构来引入一些方法，每个方法完成任务的一个步骤，这样可以使得程序变得更具有可读性。
>
> 总结：本文的重构方法在有的时候还是比较有用，但这样会造成字段的增加，同时也会带来一些维护的不便，它和“提取方法”最大的区别就是一个通过方法返回需要的数据，另一个则是通过字段来存储方法的结果值，所以在很大程度上我们都会选择“提取方法”。

软删除-EFCore在【修改操作】中实现机制。




```
 protected virtual void ApplyAbpConceptsForModifiedEntity(EntityEntry entry, long? userId,
EntityChangeReport changeReport)
        {
            SetModificationAuditProperties(entry.Entity, userId);
            if (entry.Entity is ISoftDelete &&
                entry.Entity.As<ISoftDelete>().IsDeleted)
            {
                SetDeletionAuditProperties(entry.Entity, userId);
                changeReport.ChangedEntities.Add(new EntityChangeEntry(entry.Entity,
EntityChangeType.Deleted));
            }
            else
            {
                changeReport.ChangedEntities.Add(new EntityChangeEntry(entry.Entity,
EntityChangeType.Updated));
            }
        }
```

软删除-EFCore在【删除操作】中实现机制。



```
        protected virtual void ApplyAbpConceptsForDeletedEntity(EntityEntry entry, long?
userId, EntityChangeReport changeReport)
        {
            if (IsHardDeleteEntity(entry))
            {
                changeReport.ChangedEntities.Add(new EntityChangeEntry(entry.Entity,
EntityChangeType.Deleted));
                return;
            }

            CancelDeletionForSoftDelete(entry);
            SetDeletionAuditProperties(entry.Entity, userId);
            changeReport.ChangedEntities.Add(new EntityChangeEntry(entry.Entity,
EntityChangeType.Deleted));
        }
```