**聚合**

一组相关对象的集合，数据修改的单元

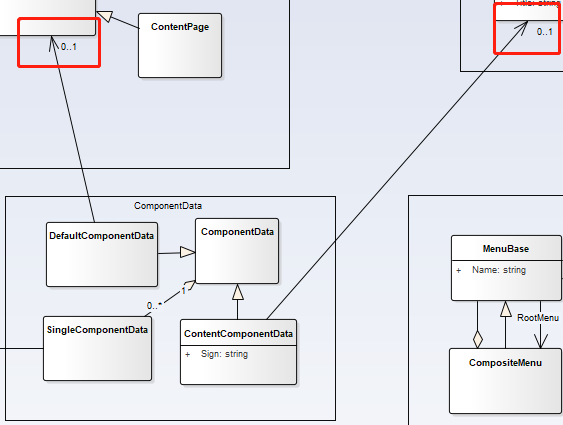
**聚合根**

访问聚和修改都从聚合根开始，聚合根是一张表，一个聚合只有一个聚合根

**聚合内部**

聚合内可以引用其他聚合根，外部对象不能引用聚合内部实体，可以将聚合内部实体引用传给外部，但只能是临时引用，聚合内部实体的声明周期由聚合根决定

如图：



聚合根是ComponentData（这里有一个疑惑？DefaultComponentData和ContentComponentData到底算不算聚合根）

**聚合根与服务的依赖关系**

服务应直接依赖与聚合根，而不要依赖与聚合内部实体

// 聚合内部实体

public class Entity1{

}

// 聚合根

public class Root {

    public Entity1 Entity1{ get; set; }

}

public class Service {

    // 不推荐这种方式

public Service(Entity1 e){}

    // 推荐使用这种方式

    public Service(Root e){

}

**聚合示例**

    // 文章，这里文章属于聚合根

    public class PageData:Entity, IHasCreationTime

    {

        public string Name { get; }

        ...

        // 文章标签

        public ICollection<Tag> Tags { get; set; }

        // 点击文章

        public void ToClick() {

            Click++;

        }

        // 文章评分

        public void ToScore(int score, int userId)

        {

            if (score < 0 || score > 10) {

                throw new UserFriendlyException("无效的评分范围");

            }

            string userIdStr = userId.ToString();

            List<string> userIds = ScoreUser == null ? new List<string>() : ScoreUser.Split("|").ToList();

            if (userIds.Any(e=>e == userIdStr)) {

                throw new UserFriendlyException("已评分，无法再次评分");

            }

            if (userIds.Count > 5000) {

                throw new UserFriendlyException("评分人数已达到上限");

            }

            double sum = Score \* ScoreNum;

            sum = sum + score;

            ScoreNum++;

            Score = sum / ScoreNum;

            userIds.Add(userIdStr);

            ScoreUser = userIds.JoinAsString("|");

        }

}

    // 标签，这里标签属于聚合内部实体

    public class Tag : Entity

    {

        public string Name { get; set; }

        public string DisplayName { get; set; }

        [ForeignKey("PageDataId")]

        public PageData PageData { get; set; }

        public int PageDataId { get; set; }

    }

**工厂**

工厂可以是工厂类，也可以是工厂方法

// 工厂类

class ProductFactory{

    public Product createProductA(){

        ....

    }

}

// 工程方法

class Product{

    public static Product createProductA(){

        ....

    }

}

- 工厂负责创建复杂的类或聚合

- 工厂的每个操作是原子的，要么成功，要么失败

- 如果工厂接收一个创建请求但无法完成创建，应该抛出异常

- 聚合的工厂方法一般放置在聚合根，也可以将一个聚合的工厂方法放置在另一个聚合内部

**工厂类创建原则**

创建满足最小规则的聚合，其他选择留给外部，如

创建User只需要用户名和密码即可，其他属性（如地址，电话等非必须属性）留给外部

class Product{

    // name 在生成后不允许改变，所以只有 get

    public string Name { get; }

public string DisplayName { get; set; }

    protected Product(string name){

        Name = name;

    }

    // name 是必须的，所以只需要 name

    public static Product createProductA(){

        // 这里为了方便，但方法内直接使用"\*"字符是不推荐的，怎么重构？请阅读重构

        return new Product("A");

    }

}

**有些时候，直接使用构造函数是最佳选择**

构造函数应该保持简单

构造函数中不要调用其它类的构造函数，那怎么办？

ClassA必须有ClassB那么， new ClassA(classB)

**仓库**

- 仓库用于持久化实体（你可以持久化到本地硬盘，或数据库等），在查询时恢复实体

- 只有聚合根才有仓库，聚合根内部通过聚合索引访问

- 通过仓库对聚合进行增删改查

- 仓库不会提交事物，事物提交在应用层

如下一个示例：

    // 页面仓库

    // PageBase 聚合根

    // EfRepository 基类包含了对实体的 增删改查 等操作

    public class PageRepository : EfRepository<PageBase, int>, IPageRepository

    {

        public PageRepository(IDbContextProvider<IEManageSystemDbContext> dbContextProvider) : base(dbContextProvider)

        {

        }

        // 获取实体

        public PageBase GetPageOfAllIncludes(string name) {

            PageBase page = Context.Set<ContentPage>()

                .Include(e => e.ContentPagePermissionCollection).ThenInclude(e => e.ContentPagePermissions)

                .FirstOrDefault(e => e.Name == name);

            if (page == null) {

                page = Context.Set<StaticPage>()

                .FirstOrDefault(e => e.Name == name);

            }

            return page;

        }

    }