NetCoder快速开发框架说明

数据库设计：

1. 用户权限设计：
   1. 概要设计：
      * 某某主体 在 某某领域 有 某某权限
      * 主体可以是用户，可以是角色，也可以是一个部门
      * 领域可以是一个模块，可以是一个页面，也可以是页面上的按钮
      * 权限可以是“可见”/“只读”/“可用”/不可用，因此用户和角色的权限不再只是并集，还需判断交集中的权限是否相同。
      * 这样的设计不仅可以给在角色下的用户，添加新的权限，也可以禁用用户在角色下的权限。
      * 总而言之，其实就是Who、What、How的问题
   2. 详细设计：
      * 把系统的配置放在数据库上，每一个系统对应一个唯一的编码SysCode.
      * 把菜单的配置放在数据库上，每一个菜单对应一个唯一的编码MenuCode，每一个“叶节点”的菜单项对应一个页面(url), 并归属于一个系统上。
      * 把按钮的配置放在数据库上，并归属于一个菜单项上（其实就是挂在某一个页面上）。应该一个页面可能会有几个按钮组，比如说有两个列表，这两个列表都需要有“增加、修改、删除”。所以需要增加一个按钮分组的字段来区分。
      * 把菜单权限分配给用户/角色，

Master为"User"或"Role",

MasterValue为UserID或RoleID,

Access为“Menu",

AccessValue为MenuNo,

Operation为"enabled"

如果需要禁止单个用户的权限，Operation 设置为"disabled"

用户和角色的权限中，用户优先级高=》部门=》角色

* 1. 参考：<https://www.jianshu.com/p/6f65212e5c39>

1. 字典数据设计：
   1. 概要设计：
      * 存储下拉选项的数据，只有类别和详细项目两级，因此使用两张表存储。
   2. 详细设计：
      * 主表存储类别，每个类别有唯一的类别编码，类别编码尽量与使用的字段名（表名）一致。
      * 明细表存储类别下的项目，每个项目在各自的类别下都有唯一的项目编码，使用类别ID做关联。
2. 文件数据设计：
   1. 概要设计：
      * 用于存储文件信息，以及与其它对象的关联关系。
      * 某某对象 与 某某文件 有关联
      * 对象可以是任何一个需要关联文件的模块。
   2. 详细设计：
      * 主表存储文件信息，路径使用相对路径，使用每一次的上传文件的标识码做叶子文件夹的名称，保证唯一，避免文件名冲突。
      * 关联对象与文件的关系是多对多时，需要使用逻辑删除，只删除关联关系，不删除文件，为一对多时，保证上传同一文件保存不同的路径，不需要逻辑删除。
      * 添加文件关联的数据，

RelateObj为字典中的项目代码,

RelateObjID为关联对象的ID

FileID为关联的文件ID

1. 知识点：
   1. 主键，外键（EF框架对数据库表外键的影响）
   2. 索引（聚集和非聚集，联合）
   3. 存储过程，触发器，和事务
   4. 查询语句的高级用法及优化
   5. 分库分表

框架概要设计：

1. 基础设施层：此层为最底层，可以为其余所有层服务。主要提供了项目开发所需的各种帮助类，拓展类，还集成了Socket通讯框架，WCF拓展使用框架，Windows服务容器等丰富的类库都是为开发人员提供了开发中常用的功能，为快速开发提供强有力的保障。
2. 数据仓储层：这层主要为对数据库操作CRUD的简单封装，以EntityFramework为核心，采用简单工厂、抽象工厂、工厂方法、三个工厂设计模式，使开发人员进行CRUD只需要极为简单的代码即可完成。本层还提供了数据库事务的支持，更是提供了分布式事务支持，为数据库操作提供必备的保障。使用本层提供的接口，无需关心具体的数据库类型。总之，本层为开发人员对数据库的操作提供了简单高效的操作接口，可以极大的提高开发效率。
3. 实体层：这层主要为ORM框架数据库表对应的实体类，为业务逻辑层和应用层服务。
4. 业务逻辑层：本层是开发人员主要编写层，通过调用数据仓储层操作数据库，并为应用层提供所需的接口，处理具体操作的业务逻辑，可以说是最为复杂的一层。
5. 应用层：本层在本框架中表现形式为ASP.NET MVC网站，。本层为具体的应用，负责系统功能的实现。
6. 引用：
   1. System.Linq.Dynamic，EF6.0，System.ValueTuple，Microsoft.AspNet.WebPages

System.Configuration.Configuration，Aspose.Cells，SharpZipLib，

StackExchange.Redis，Oracle.ManagedDataAccess，Npgsql，MySql.Data，

Autofac，Castle.Core，System.ServiceModel(SynchronizedCollection).

Nlog，Elasticsearch.Net，System.ComponentModel.DataAnnotations(数据库映射属性)，

AutoMapper，Autofac.Extras.DynamicProxy，Autofac.WebApi2，Autofac.Mvc5，DotLiquid

框架详细设计：

1. 基础设施层
   1. 拓展类
      * Byte拓展：主要为字节和字节数组与Int/String/Base64String/二进制字符串/16进制字符串/ASCII之间，使用特定字符编码的转换，以及将一个序列化后的byte[]数组还原，取异或值。
      * Int扩展：转换为字节数组，将JS的GetTime值转为DateTime类型
      * DateTime扩展：获取某一日期是该年中的第几周，获取JS的GetTime值，以及获取默认时间1970-01-01。
      * Object扩展：判断NULL或空，对象序列化返回字节数组，深复制，以 及序列化为JSON/XML字符串，更改实体类型，获取对象属性值，初始化对象的日期类型的属性为时间默认值。数值类型转换。
      * String扩展：二进制/16进制/Base64/String字符串与byte[]/Int/double/datetime的转换，ASCII转换为字节数据，使用指定字符编码转换为Base64String /Md5加密字符串/SHA1哈希加密/MurmurHash2-Redis，Json/XML字符串反序列化为为对象，Json转为Jarray/Jobject/DataTable/List/实体类，转换为IPEndPoint和枚举类型，以及首字符大/小写。
      * DataTable扩展：转为List集合和CSV字符串。使用IL.Emit的两种方式。
      * Type扩展：类型转换的公共方法，转换为对应的Nullable'T'类型。
      * Stream扩展：转为字符串和字节数组。
      * Expando扩展：属性的操作，转为DataTable。
      * IEnumerable扩展：序列中的替换，复制，转换（DataTable，List），扩展ForEach方法，序列连接对象，分页。生成SQL条件。
      * IQueryable扩展：分页，LinqKit扩展，获取SQL语句/ObjectQuery，表达式的删除和获取(skip/take/order)，切换DBContext/数据源，动态函数Min/Max/Sum等。
      * Expression扩展：重新组合继承属性的选择表达式树，And/Or表达式连接，及Invoke方法。
      * Delete扩展：异步执行下一个方法。
      * ActionExcutionContext扩展：MVC中判断过滤器上下文是否包含某特性。
   2. 帮助类
      * OfficeHelper：Excel的读取和导出。
      * ConfigHelper：获取配置文件的AppSettings和ConnectionStrings的值。
      * DelegateHelper：异步执行下一个方法。
      * DbSearchHelper：分组统计查询（行转列），反射获取IQueryable对象。
      * EmitHelper：获取泛型对象创建者。
      * ExceptionHelper：获取格式化的异常信息。
      * FileHelper：操作文件目录，文件的读取和写入，以及合并。
      * FileZipHelper：zip文件减压。
      * HTTP Helper：Get/Post请求，获取请求参数，从URL获取图片/HTML文档。
      * IdHelper：生成唯一主键（雪花ID）。
      * ImgHelper：从文件/base64字符串/Url中读取图片，图片压缩，图片转换为Base64String或Url，图片上传返回地址。
      * IpHelper：获取本机IP，可用端口判断和获取。
      * LinqHelper：初始化条件为true/false的表达式。
      * PathHelper：获取网站虚拟路径对应的URL和绝对路径，相对/绝对路径转换
      * RandomHelper：随机数和列表的随机值。
      * TimerHelper：间隔/等待多长时间循环执行。
      * TreeHelper：生成树结构数据，获取所有子节点。
      * TypeBuilderHelper：创建类型。
      * IOCHelper：（控制反转）注册类型，获取对象。
      * SessionHelper：缓存的获取和删除。
   3. 系统缓存
      * HttpRuntime和Redis：设置和获取，过期时间设置。
   4. 数据访问
      * DbFactoryHelper：使用控制反转，生成不同类型数据库的操作帮助类
      * DbHelper：生成实体文件（数据库类型转为C#类型）
      * DbObjectHelper：提供数据的操作对象，数据库类型和字符串间转换。
   5. AOP
      * 使用Castle.Core动态代理，注册拦截器和业务组件，并关联。
   6. DI
      * 使用Autofac管理依赖注入，通过注册及业务类的依赖标识。
2. 数据仓储层：
   1. DbContext：基于EF的数据库对象的封装和类型工厂。
   2. Repository：包含了增、删、改、查、事物控制等数据库常用核心操作，以及SqlServerRepository、MySqlRepository和PostgreSqlRepository分别实现具体数据库对应的数据仓储。
3. 数据实体层：
   1. 数据库实体映射。
4. 业务逻辑层：
   1. 业务逻辑拦截属性的实现：操作日志和数据重复验证。
   2. 业务逻辑基类：实现数据仓储的调用（增删改查，事务操作），并提供数据仓储类型的属性Service，支持联表操作。并提供通用业务逻辑返回数据的方法。
   3. 日志管理：NLog日志自定义。
   4. 代码生成：根据数据库表结构，生成实体类，业务层，控制器，视图的代码。
5. 应用展示层：
   1. 使用Admin-Lte前端框架，安装Bower包管理，cmd切换磁盘，直接E:，然后cd.
   2. Bower使用git管理包版本，需要翻墙或者代理。不推荐使用。
   3. 快速开发主页和和登录页：MVC模板页，Admin-Lte样式，动态生成左侧目录树，frame内容。菜单的树结构和列表结构转换。
   4. 代码生成页面和数据库连接管理页面：使用控制器Controller管理数据和视图。
   5. 过滤器Filter：
      * 登录身份验证：使用cookie和cache存储信息， 来管理session，通过重写MVC的全局授权过滤器，或者重写筛选器特性的基类，来控制访问。
      * Token和签名：防抵赖、防伪造、防重复调用。
        1. <https://blog.csdn.net/wnvalentin/article/details/89854980>
        2. <https://www.bbsmax.com/A/n2d9YKZ0dD/>
        3. <https://www.cnblogs.com/Leo_wl/p/5982927.html>
      * 权限验证：接口的权限校验。（页面和操作）
      * 异常处理：重写异常处理过滤器HandlerErrorAttribute，记录日志。
   6. Web缓存：使用Cokie，Cache存储。其它方案：session，nosql，
      * 区别：<https://blog.csdn.net/mjx258258001/article/details/104471532>
   7. 初始化依赖注入和类型映射：AutoFac和AutoMapper
      * 使用AutoFac管理依赖注入，支持属性和构造函数的注入，添加MVC和WebAPI的容器设置。
      * 注册AOP和AutoMapper。
   8. EF优化：启动预热，执行一次数据获取的操作。
      * 原因：对于在应用程序中定义的每个DbContext类型，在首次使用时，Entity Framework都会根据数据库中的信息在内存生成一个映射视图（mapping views），而这个操作非常耗时。
      * 参考：<https://www.cnblogs.com/xcsn/p/9261898.html>
   9. DotLiquid模板引擎：
      * 初始化视图模板引擎，注册视图中注入的对象。