**基础框架设计说明**

# 概要设计

## 背景

项目多采用快速开发完成功能，目前大屏展示项目大多基于数据查询实现，很大一部分简单的增删改查功能需要重复写相同逻辑的代码。为了解决此问题，本框架拟设计出一种可以实现开箱即用的通用实现，简单的查询只需要在业务清晰，表设计完成之后即可对表自动完成简单的业务处理。业务只需要再单独处理复杂的业务场景；同时本框架集成了统一的权限处理体系，业务开发无需过多关注权限的处理及实现，大大节省了开发时间；让后台工作重心集中在复杂的业务实现上。

## 设计说明

本设计主要分为公共通用工具的封装、代码生成器、权限设计体系三部分，分别对应不同的解决场景。

框架选型主要以spring、springmvc、mybatis、mybatis-plus 为主，通过springboot 快速集成相应的框架，搭配一些实用工具包以拓展工具丰富性。同时统一管理依赖版本。

### 公共通用工具

公共通用工具目的是实现统一的前后端交互，保持风格统一。包含统一数据返回模型；项目中定义的一些基本的概念模型如用户、部门等；统一定义异常封装及返回，保证异常情况下返回数据的统一性；统一定义基础配置信息，如swagger的配置，web的放行策略、跨域等处理，持久层框架mybati/plus的配置等。

### 代码生成

统一按照既定的标准生成基础的后台代码，实现了业务层的基础搭建，保证生成后可以直接实现业务使用，不用再从0搭建模块代码，只需要从1开始后续开发。每个表可以单独生成代码，也可以多个表甚至全部表同时生成，实现开箱即用。

### 权限体系

统一了权限的实现及统一维护管理，实现业务权限的统一管控，将业务系统的权限用统一的模块管理起来，既实现了项目权限的管理的必要性，同时也可以防止各种业务模块中出现不同的权限实现方案，减少了系统权限管理的难度。

# 详细设计

详细设计中主要详细解释业务中的实现思路、逻辑，以及权限体系的具体设计流程及代码接入及实现。

## 公共通用工具

### 统一框架版本管理

### 统一返回模型

定义BaseResponse模型，统一返回，在基于前后端分离的项目中，统一返回的必要性很高，可以极大减小前端处理的复杂度。

|  |
| --- |
| @Data  public class BaseResponse**<**T**>** **{**  /\*\*返回状态码\*/  private Integer code **;**  /\*\*应答消息\*/  private String message **;**  /\*\*返回业务数据\*/  private T data**;**  private BaseResponse**(){**  **this.**code **=** Constants**.**SUCCESS\_CODE**;**  **this.**message **=** Constants**.**SUCCESS\_MESSAGE**;**  **}**  public static BaseResponse**<**Void**>** success**(){**  **return** **new** BaseResponse**<>();**  **}**  public static **<**T**>** BaseResponse**<**T**>** success**(**T data**){**  BaseResponse**<**T**>** baseResponse **=** **new** BaseResponse**<>();**  baseResponse**.**data**=**data**;**  **return** baseResponse**;**  **}**  public static BaseResponse **<**Void**>** failed**(**Integer code **,**String message**){**  BaseResponse**<**Void**>** baseResponse **=** **new** BaseResponse**<>();**  baseResponse**.**code**=**code**;**  baseResponse**.**message**=**message**;**  **return** baseResponse**;**  **}**  public static BaseResponse **<**Void**>** failed**(**String message**){**  **return** failed**(**Constants**.**FAIL\_CODE**,** message**);**  **}**  **}** |

返回以静态数据返回，泛型即为传入数据类型和数据，统一返回数据体之后前端主要根据code即可判定业务成功状态，成功则通过data取值。此方式前端也可做统一的异常提示，减小前端处理异常工作量。

### 统一异常处理

项目统一异常处理，业务中的异常可直接外抛，无需更多的处理，使得代码简洁。异常统一在springmvc提供的异常增强处理中处理，处理的同时将其转换为统一的返回对象。

### 统一请求配置管理

统一请求配置预先配置好项目中共有的web配置，一般项目中的配置大体上是固定的，通过使用相同的配置使项目的运行环境保持一致，主要配置项目中需要使用到请求拦截器的地方，如做一些通用的在到业务层之前的请求增强，跨域处理，放行策略等。

### 统一数据库持久层配置

预先定义好数据库持久层使用的连接工具，包括mysql-connector,mybatis,mybatis-plus，同时定义好mybatis-plus的分页插件，逻辑删除插件，枚举，连接池，sql打印等。

## 代码生成

代码生成部分依赖于mybatis-plus提供的代码生成器，同时依赖velocity模板引擎，通过定义好引擎的模板，自动生成从controller 至 mapper的代码，实现0代码即可使用简单的业务逻辑。

代码生成主要包括两部分，第一部分是代码生成的逻辑，第二部分是相关配置，新建模板的时候需要少量的配置。

### 约定说明

代码生成器的配置约定统一。

其中生成的结构代码均在工程路径+基础包+模块下，对应的结构为controller包、service包、mapper包、entity包，其命名方式为表名转驼峰后拼接上Controller或Service 或Mapper，Entity为表模型，与表保持一致。

Maper对应的xml 统一在resources下的mapper/模块名/下，其命名方式为表名转驼峰拼接上Mapper.xml，在启动时mybatis-plus自动扫描classpath:mapper/目录下的\*Mapper.xml文件。

### 配置说明

代码生成的配置全部定义在CodeGenerator的静态内部类Config中，代码生成前需要根据实际使用定义部分的配置，以下是对应的配置字段的解释：

AUTHOR: 作者，即包名上对应的开发人员。

ABSOLUTE\_PATH： 绝对路径，定义了绝对路径的话代码生成的时候会自动生成到对应的目录中，路径统一指项目src目录上一级。如果此选项为空，则以当前的工程路径为准。

BASE\_PACKAGE：基础包名，即项目中代码生成的包的基本的路径，如基础框架的com.chinaunicom.cn.base 即为基础包名。

MODULE\_NAME：模块名，即项目中的模块，一般同一类似的业务功能单独定义一个模块，也可以统一指定一个模块（如web模块）。

APPLICATION\_CONFIG\_FILE：配置文件的路径，配置文件主要是需要jdbc连接，供连接数据库使用。一般以dev开发环境中的为准，在开发时生成代码，文件的格式如application-dev.properties 或application-dev.yml 格式与spring的配置保持统一，只需要spring.datasource.url，spring.datasource.username，spring.datasource.password这三个参数。如果需要手动定义数据库配置，不与开发环境保持一致的话，可以自行修改代码getDateSourceConfigBuilder() 方法的返回，直接return new DataSourceConfig.Builder(url,username,password) 即可。

CONTAINS\_TABLE\_PREFIX：指定需要生成的表前缀，按照模糊前匹配原则，配置了此选项则表示只生成对应前缀的表，其余不匹配的则不生成；若此项参数为空，则表示所有表都需要生成。

IGNORE\_TABLE\_PREFIX：指定需要忽略的表，在指定得到的表结果后在结果集中再次忽略掉需要过滤的表，更精准的控制需要生成的表。

注意：代码生成会覆盖掉原来的已经生成了的代码，若需要忽略已经生成的文件，建议指定忽略的表，也可以修改createAutoGenerator方法中的配置，建议不要用后者；

### 生成器说明

1.加载数据源

生成器首先会先根据配置加载数据源，配置默认从Config定义的文件名加载文件，文件的格式要求为.properties或.yml，程序首先加载properties文件，在未找到.properties的情况下再加载.yml文件，文件加载符合spring的规范，主要找的参数为spring.database.url、spring.database.username、spring.database.password三个参数，即应用程序中定义的spring的数据源。解析后统一保存成DataSourceConfig.Builder 对象。

2.获取需要生成配置的表名

1) 通过DataSourceConfig.Builder得到DataSourceConfig配置，再通过DataSourceConfig创建JDBC连接；

2) 通过url中的数据解析得到表的schame，查询mysql的information\_schema.tables得到所有的表；

3) 拿到表名之后再匹配符合配置要求的表，再过滤掉不需要生成的表

3. 创建配置

根据mybatisl-plus提供的插件，配置FastAutoGenerator ，其中已经按照行业约定开发规范定义好了大部分的模板引擎的配置和方法，并且自定义Config中的配置也会加载到对应的配置中。

4. 执行代码生成

## 权限体系

本框架搭建权限体系拟从权限主体设计、应用权限转换及使用、权限存储三个维度设计整套权限体系，拟完成一个统一的权限处理体系。在权限体系中维护绝大部分的逻辑，业务只需要简单的添加对应的注解即可实现对权限的校验及配置，同时权限注解配合代码生成器，业务代码在生成后就自动加上了权限控制。而且业务逻辑与权限解耦，业务开发时不用在关注权限相关的，只需关注自己的业务逻辑即可。

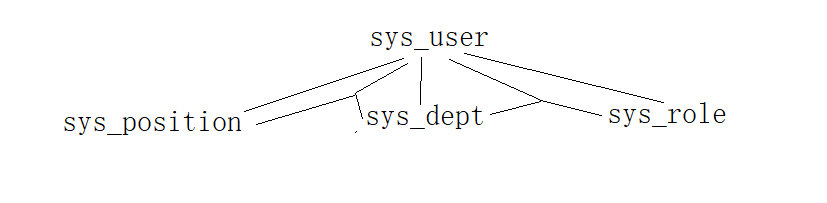
区分三个维度的主要母的是实现了权限体系的职责分离，并预留拓展空间。权限主体的职责是维护用户的整个画像，包括用户的机构、岗位、角色等；应用权限转换及使用主要是完成将用户的一系列画像转换为统一的标准的权限结构或反之，并在业务中能方便的接入、校验、判定；权限存储主要是针对业务对于权限的存储关联，菜单与权限的关联，并提供在数据库持久化的能力。

### 权限主体设计

市面上绝大部分的权限体系均基于RABC的权限设计思想，实现 用户-角色-权限-资源的对应关系。但是这种设计具有很大的局限性。在大企业组织环境下，基于机构、岗位、角色的权限控制是必然存在的，传统的RABC只能满足用户-角色-资源的对应关系。

本设计采用树形组织架构的模型设计，可以无限递归部门，并配置对应的岗位，符合正常的树形架构，同时支持添加相应的角色。

用户一般在组织架构的维度为从属于某一个部门，在部门中有对应的岗位，但是也可以拓展用户拥有其他部门的岗位，或者用户从属于多个部门（不建议），用户可以配置角色，也可以配置某个部门下的角色。因此用户的画像可以为：部门、部门+岗位、部门+角色的形式， 可以满足大部分权限体系的使用，如图：



为了便于使用，减小复杂度，暂时只设计了部门、岗位、角色三个维度的权限体系。若有业务需求，还可以进行拓展。

角色可以增加角色组的概念，用户配置角色的时候可以配置一组角色，减小维护的成本。

若需要其他维度的概念，也可以在此基础上进行拓展。

### 应用权限设计

传统的应用权限设计采用RABC的权限思想，权限的具体维护通过资源关联菜单实现，除了上述说的单一拓展性不强外，还有的问题为只能实现菜单级别的权限设计，对于具体的业务权限，还需在每个业务中单独维护一套权限关联上RABC上的概念，而且校验时也需要每个接口主动关联相应的业务代码，增加了核心业务的开发时间，且校验权限时组织架构冗余在权限校验中，非常不符合服务分离、应用解耦的思想，增加了系统的复杂度。

本系统设计借用了SpirngSecurity的权限思想，将权限统一成既定格式的权限字符串，在业务使用校验时全部使用字符串形式的校验，而用户的权限来源则来源于权限主体，将用户的画像转换为对应的权限字符串，实现了用户的画像和应用权限的分离。此举将权限的配置与用户权限主体解耦，使得权限在应用中更为清晰。

#### 数据库权限结构

权限配置主要包括菜单权限控制和数据权限控制。

1. 菜单权限控制：利用一张菜单、权限字符串的关联表实现菜单、按钮级别的权限控制，如下图：

菜单表sys\_menu：



菜单权限表sys\_menu\_auth：



菜单权限的控制具有细粒度的控制，已经实现了根据用户、机构、岗位、角色的多维度权限配置，通过统一的权限配置体对象配置每条菜单的权限。菜单表与菜单权限表的对应关系为菜单表的id与菜单权限表的menu\_id一对多关联，权限配置在权限表的authority字段中。

同时菜单权限还实现了下级菜单以上级配置的菜单权限为准，实现了权限简化，更符合业务需求。其配置为菜单表的auth\_type类型，类型判断为1则基于上级菜单，类型为2则基于本级菜单权限。基于上级的权限也可以递归网上追溯，可以一直往上级菜单追溯。

2）业务权限控制：利用一张业务id、权限字符串关联表实现权限与业务权限的关联控制，如下图：



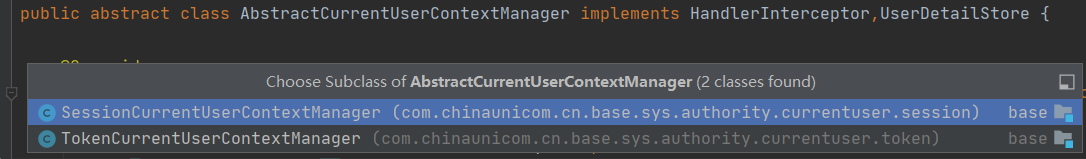
业务的权限关联关系为业务的主键id关联应用权限表的business\_id，实现数据权限的控制，设置的权限保存在authority对象中，与菜单的权限结构保持统一。

#### 应用权限实现

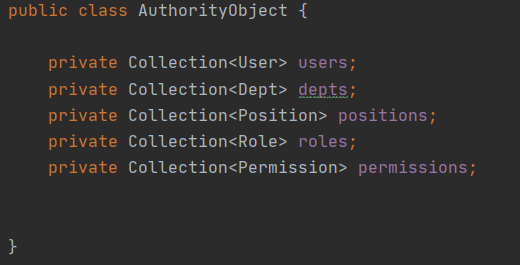
应用权限的实现主要包括登录用的权限保存、业务权限的转换、用户和业务数据的匹配关系三部分。

1. 登录权限实现：在用户登录时先将用户的部门、岗位、角色等用户画像查询出来，通过转换类转换成权限字符串，然后存储在用户凭证中，每个用户即拥有了自己的权限字符串集合。当用户发起请求时，首先通过拦截器/过滤器在保存的凭证中获取到用户的上下文数据，然后再校验用户权限。

用户凭证保存已经有两个实现，一个是基于session的实现，当登录完成后，后台保存到session中，用户发起请求时，直接从session中获取。第二个实现是基于token的实现，用户登录后生成token，并将用户画像和token保存在内存/redis中(可以扩展)。如下图：



1. 业务权限转换：在业务数据添加时，同时带上权限对象，调用权限设置接口添加权限至对应的表中。前端传入的权限接口如下图：



在权限设置对象中定义了可配置的用户、机构、岗位、角色等（还可以再此基础上做拓展，如数据管辖地域等），通过统一的转换对象转换为数据库统一定义的权限结构，如下图：



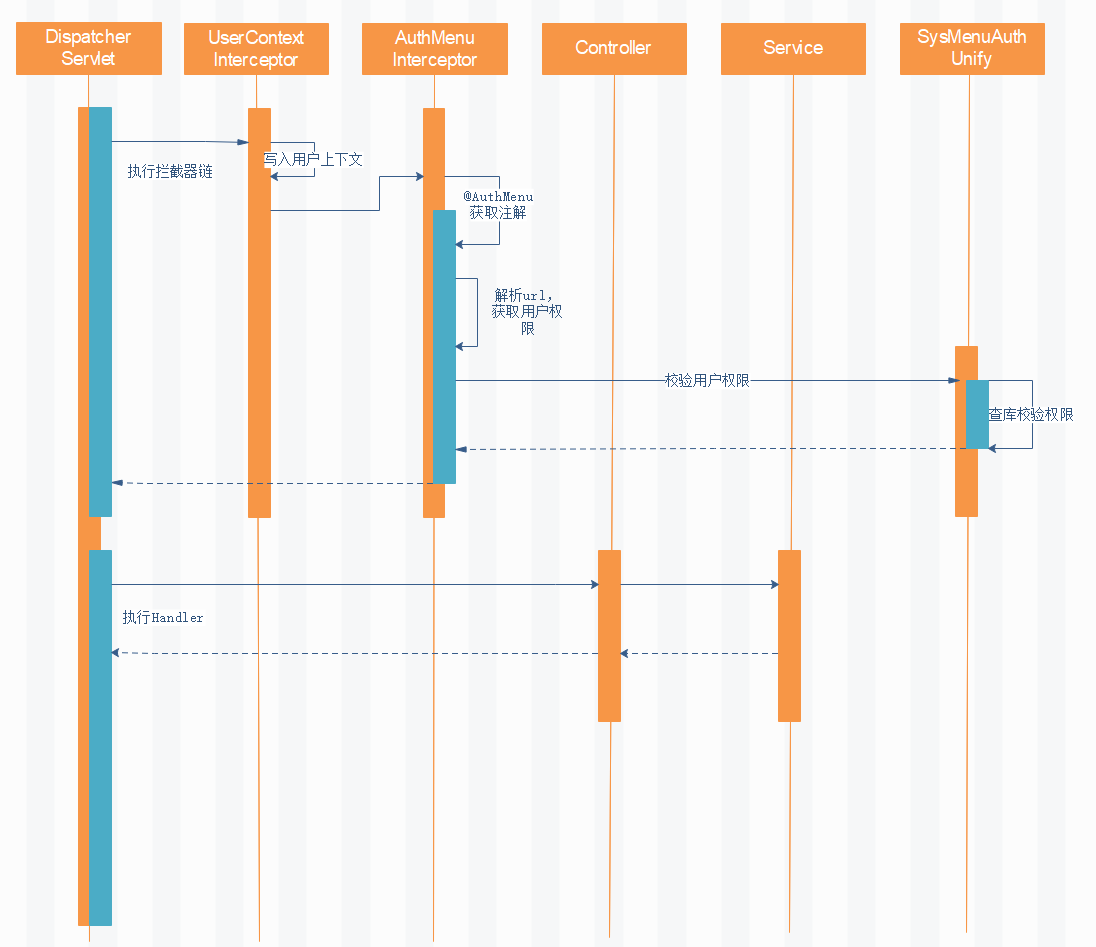
这样，前端直观的权限结构体和数据库统一定义的权限格式实现了相互转换。

1. 权限校验：
2. 菜单列表

菜单列表的权限实际上是菜单的可见性，通过关联菜单表，菜单权限表，再将用户的上下文中的权限字符串带入关联即可得到用户的菜单权限，不加以深入说明。

1. 菜单按钮权限校验

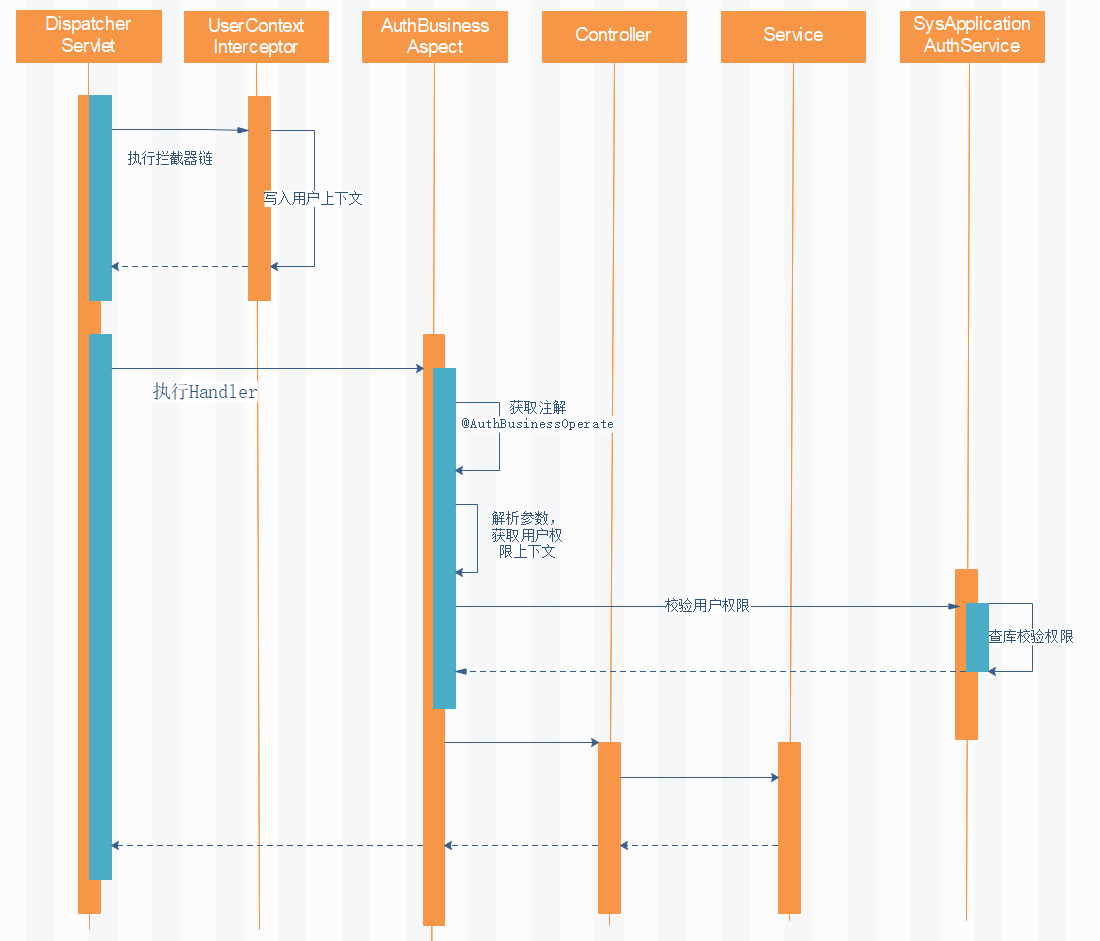
菜单按钮的请求访问权限通过SpringMVC的拦截器实现，在请求到达拦截器中时，业务流程如下图：



首先校验controller中是否有@AuthMenu注解，将用户的权限从上下文菜单中获取，再根据请求获取到url地址，然后将两者带入菜单和菜单权限联查数据校验权限。若查询有数据则校验通过，继续执行后续逻辑，若校验不通过则直接异常返回无权限数据。业务逻辑只需要一个注解即可完成权限的校验，完全不用关系具体的权限逻辑，实现解耦。

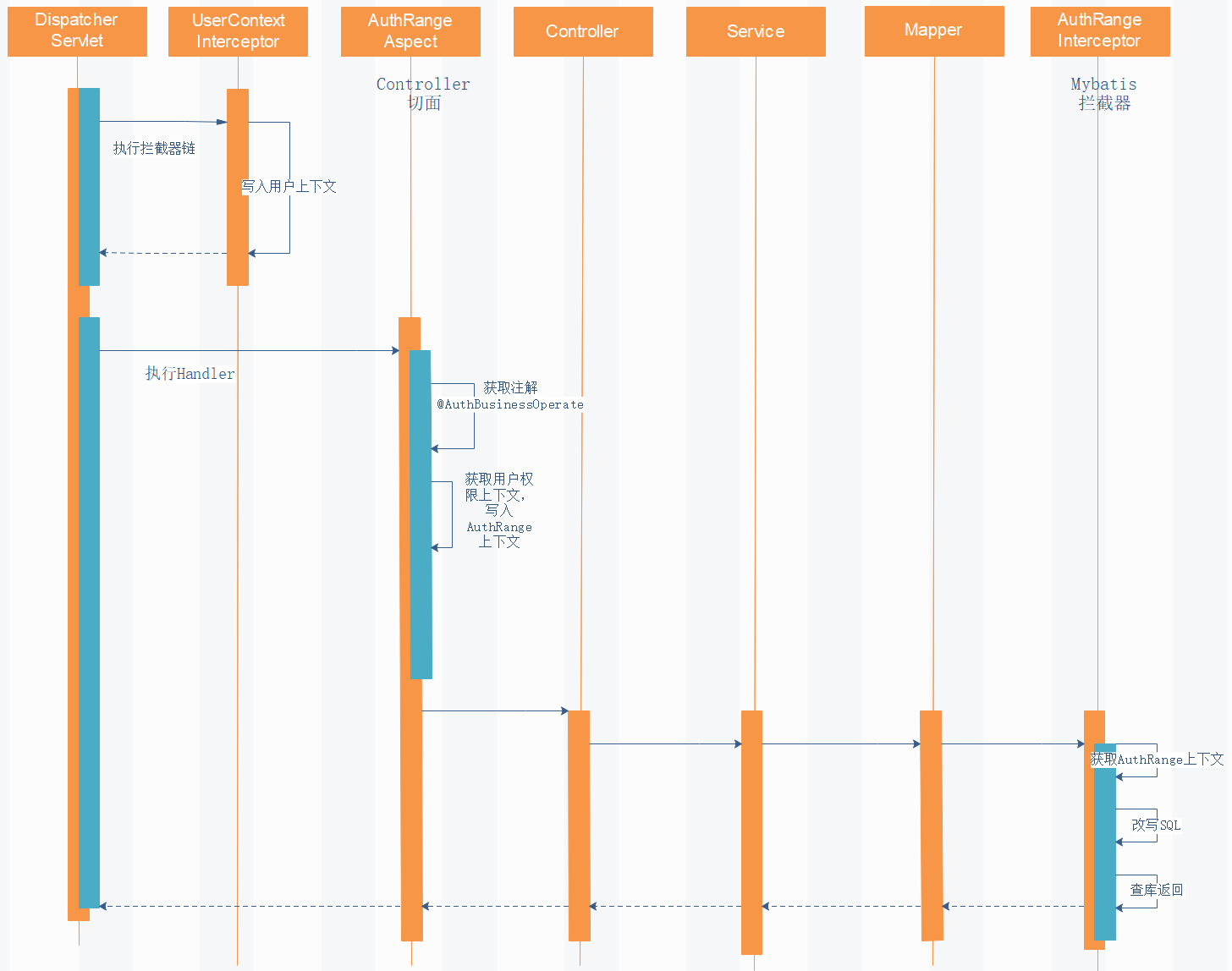
1. 数据权限校验

在校验了菜单请求权限后，执行数据执行权限校验，如下图：



数据执行权限通过controller切面实现，在数据到达controller时增加前置校验增强，拦截请求。在请求中解析请求访问的主键id，拿到用户上下文，再查询数据权限关联表查询是否有对应的数据权限，若有则验证通过，若没有则直接抛异常结束请求，实现统一的数据权限校验通知。

1. 数据可见权限校验：通过mybatis的拦截器实现SQL改写，如下图：



当请求到来时，首先通过菜单权限校验接口是否可访问，然后访问业务数据。数据可见权限是通过SQL过滤筛选的，通过在持久层框架mybatis上改写最终的访问SQL，将业务id关联业务权限表并进行数据过滤，从而实现数据权限的控制。

### 权限设计总结

本设计实现了菜单权限细粒度的控制，菜单的逻辑除了直接设置本级菜单使用，同时也可指定本级的菜单权限继承父菜单权限，可以有效避免每个菜单均配置权限的情况，减小了配置的数据量，也更符合业务场景。

本设计在应用中通过统一的权限结构对象和统一的权限转换工具实现权限对象和权限字符串的互相转换。在匹配权限和存库时使用权限字符串结构，在应用中返回给前端或前端入参时使用统一权限对象，前端就可以直观的查询到某数据或菜单对应的权限结构。

# 接口说明

## 菜单权限

### 菜单权限校验

访问的菜单先通过@AuthMenu注解拦截，通过spring-mvc的拦截器获取到登录人的权限数据，再匹配菜单权限校验是否有访问权限。需要校验菜单权限的直接在controller的方法上加注解即可。注解如下：

|  |
| --- |
| @Documented  @Retention**(**RetentionPolicy**.**RUNTIME**)**  @Target**(**ElementType**.**METHOD**)**  public @interface AuthMenu **{**  /\*\*  \* 路径，为空则以输入Controller的RequestMapping为准  \* **@return**  \*/  @AliasFor**(**"value"**)**  String url**()** **default** ""**;**  @AliasFor**(**"url"**)**  String value**()** **default** ""**;**  **}** |

url: 菜单链接，菜单表中对应的链接，可以不填，默认为访问的requestMapping对应的路径。

Value: 作用同url；

### 查询登录人拥有的菜单列表

接口位于com.chinaunicom.cn.base.sys.controller.AuthorityController，接口路径/authority/getVisibleMenus，get请求方式。此接口查询当前登录人有权限访问的接口，当用户没登录或未配置权限时，返回默认的N-NONE权限。此接口不做权限校验。

### 修改菜单信息

接口位于com.chinaunicom.cn.base.sys.controller.AuthorityController，接口路径/authority/updateMenu，post请求方式。此接口会先通过@AuthMenu 注解校验菜单的访问权限，因此需要先配置好修改菜单链接的访问权限，有权限时则会修改菜单的信息。

### 编辑菜单权限

接口位于com.chinaunicom.cn.base.sys.controller.AuthorityController，接口路径/authority/editMenuAuthorities，post请求方式。此接口会先通过@AuthMenu 注解校验菜单的访问权限，因此需要先配置好修改菜单权限的权限。注意此处未做进一步的控制，修改权限时不要将所有的修改菜单权限的权限都设置为空，否则无法再通过程序设置菜单权限了。此处可做优化(如超级管理员默认就有此接口访问权限等...)。

## 业务权限

### 业务数据范围查询校验

业务数据范围查询通过@DataRange注解，mybatis拦截实现。在查询mapper前添加@DataRange注解（一般在列表查询接口的Service层添加），添加上注解后会在持久层查询SQL时拼接上范围查询SQL，业务中不用再关注登录人权限的差异。

注解如下：

|  |
| --- |
| /\*\*  \* @descrpition 查询权限范围注解  \* 在mapper拦截 ，用于数据权限列表查询  \* 例如 拼接语句：  \* select \* from ${business\_table} where  \* ${business\_table}.{business\_id} in  \* ( select business\_id from sys\_application\_auth  \* where application=#{application} and operate=#{operate} and authority in ( #{用户权限} )  \* )  \* 拼接 where后面部分，where前半部分为业务定义的sql，本增强只是在后加上 and 的权限查询语句，避免业务中都需要主动关联权限表匹配，  \* 实现逻辑解耦  \* where后有条件也不用担心，会自动加上and  \* **@author** kewen  \* **@since** 2022-11-23 10:52  \*/  @Documented  @Retention**(**RetentionPolicy**.**RUNTIME**)**  @Target**(**ElementType**.**METHOD**)**  public @interface DataRange **{**  /\*\*  \* 模块  \* **@return**  \*/  String module**()** **;**  /\*\*  \* 表别名，用于拼接 t.id  \* **@return**  \*/  String tableAlias**()** **default** ""**;**  /\*\*  \* 业务主键column名 用于拼接 t.id  \* **@return**  \*/  String businessColumn**()** **default** "id"**;**  /\*\*  \* 默认统一的  \* **@return** 返回操作类型  \*/  String operate**()** **default** "unified"**;**  /\*\*  \* 条件匹配方式 in/exists  \* 关联原则 小表驱动大表  \* 默认通过in的方式，当权限表中数据大时应该采用exists方式  \* **@return**  \*/  MatchMethod matchMethod**()** **default** MatchMethod**.**IN**;**  **}** |

module()：模块名。

tableAlias()：表别名，即关联查询时的主表(对应业务主键的表)的表别名，当只有一张表时无需指定。

businessColumn(): 业务主键column名，用于权限关联主键id，默认为id。

operate()：操作类型，如查看操作，编辑操作...默认为unified 即为不区分操作类型，只要能列表查询到就能读取和操作。若需要区分读权限或写权限则需要定义不同的操作。

matchMethod()：匹配方式，IN或者Exists，默认通过IN的方式关联，当业务权限表数据数量级远远大于业务数据时，应当使用Exists方式。遵从小表驱动大表的方式。

### 业务权限校验

业务数据权限校验通过@DataCheck注解，切面拦截实现。在有注解的方法上通过切面先校验是否有数据对应权限，然后再执行业务的操作。

注解如下：

|  |
| --- |
| /\*\*  \* @descrpition 业务操作权限 加在controller切面  \* select business\_id from sys\_application\_auth  \* where application=#{application} and operate=#{operate} and business\_id=#{businessId} and authority in ( #{用户权限} )  \* limit 1  \* businessId需要关联获取 {@link ApplicationBusiness}  \* **@author** kewen  \* **@since** 2022-11-23 11:55  \*/  @Documented  @Retention**(**RetentionPolicy**.**RUNTIME**)**  @Target**(**ElementType**.**METHOD**)**  public @interface DataCheck **{**  /\*\*  \* 模块ID  \* **@return** 模块ID  \*/  String module**()** **;**  /\*\*  \* 操作  \* 默认统一的 对应authority\_reference表中的 operate字段，  \* 因为数据库不留空，若没有对操作的要求则默认unified  \* 可以为 unified modify update delete 或者其他自定义的操作  \* **@return** 返回操作类型  \*/  String operate**()** **default** "unified"**;**  **}** |

module： 模块名，必须指定；

operate：操作类型，如查看操作，编辑操作...默认为unified 即为不区分操作类型，只要能列表查询到就能读取和操作。若需要区分读权限或写权限则需要定义不同的操作

除了需要指定注解外，还需要其中一个参数实现接口ApplicationBusiness ，接口参数如下：

|  |
| --- |
| public interface ApplicationBusiness **{**  /\*\*  \* 获取到关联的ID  \* **@return** 业务关联ID，业务具体事项的主键  \*/  Integer getBusinessId**();**  **}** |

getBusinessId()：获取业务主键id；即需要校验的业务数据主键id。

对于接口ApplicationBusiness ，已经有一个默认实现DefaultApplicationBusiness，可以装配此对象或者自己实现。

### 业务权限编辑

编辑业务权限暴露了两个入口，第一个入口为直接提供的对外访问服务，供给前端直接使用。第二个接口为通过注解@AuthDataEdit的方式集成在业务系统中，通过切面实现。

#### 通过web接口方式

接口位于com.chinaunicom.cn.base.sys.controller.AuthorityController，接口路径/authority/editMenuAuthorities，post请求方式。此接口会先通过@AuthMenu 注解校验菜单的访问权限，因此需要先配置好修改菜单权限的权限。

接口请求参数如下：

|  |
| --- |
| @Data  public class BusinessAuthorityEditReq **{**  @NotNull  private Integer businessId**;**  @NotBlank  private String module **;**  private String operate **=**"unified"**;**  @NotNull  private AuthorityObject authority**;**  **}** |

businessId: 业务系统的主键id，用于校验业务时的关联关系；

module：模块名称，业务系统的从属模块；

operate : 操作类型，如查看操作，编辑操作...默认为unified 即为不区分操作类型，只要能列表查询到就能读取和操作。若需要区分读权限或写权限则需要定义不同的操作。

Authority：权限结构体，对应设置的权限。

#### 通过注解@AuthDataEdit方式

业务系统在需要修改对应业务权限时添加注解@AuthDataEdit，在有注解的方法上通过切面先修改好业务权限，然后再执行业务的操作。此方式需要注意事务，在修改业务权限的时候会开启一个事务，因此后续的修改中应该不再使用事务或使用事务传播特性为required。

注解@AuthDataEdit接受的参数为

|  |
| --- |
| @Documented  @Retention**(**RetentionPolicy**.**RUNTIME**)**  @Target**(**ElementType**.**METHOD**)**  public @interface AuthDataEdit **{**  /\*\*  \* 模块ID  \* **@return** 模块ID  \*/  String module**()** **;**  /\*\*  \* 操作  \* 默认统一的 对应authority\_reference表中的 operate字段，  \* 因为数据库不留空，若没有对操作的要求则默认unified  \* 可以为 unified modify update delete 或者其他自定义的操作  \* **@return** 返回操作类型  \*/  String operate**()** **default** "unified"**;**  **}** |

module： 模块名，必须指定；

operate： 操作类型，同上；

除了需要指定注解外，还需要其中一个参数实现接口AuthDataEditBusiness

|  |
| --- |
| public interface AuthDataEditBusiness **{**  /\*\*  \* 获取业务ID  \* **@return**  \*/  Integer getBusinessId**();**  /\*\*  \* 获取权限对象  \* **@return**  \*/  AuthorityObject getAuthorityObject**();**  **}** |

getBusinessId()： 获取业务id，即需要修改对应权限的业务主键；

getAuthorityObject()：获取传入的权限对象；可以不传则删除所有权限；

对于接口AuthDataEditBusiness，已经有一个默认实现DefaultAuthDataEditBusiness，可以装配此对象或者自己实现。