## 服务划分

## 文档结构

## 类库

* Furion.Extras.Admin.NET：基础框架层
* QMS.Application.Issues：问题管理服务层（业务代码主要编写层）
* QMS.Application.System：基础数据管理服务层（业务代码主要编写层）
* QMS.Core：核心层（实体，仓储，其他核心代码）
* QMS.Database.Migrations：EFCore 架构迁移文件层
* QMS.EntityFramework.Core：EF Core 配置层
* QMS.Web.Core：Web 核心层（存放 Web 公共代码，如 过滤器、中间件、Web Helpers 等）
* QMS.Web.Entry：Web 入口层/启动层

## 接口命名规范

## 接口分组规范

接口多分组是一个系统中必备功能，我们可以将系统划分为多个模块，每个模块都独立的 api 配置。分组后的Swagger文档，会按照分组进行展示接口。如问题管理服务、基础数据服务

## 接口路径规范[​](https://dotnetchina.gitee.io/furion/docs/specification-document#632-%E6%8E%A5%E5%8F%A3%E8%B7%AF%E5%BE%84%E8%A7%84%E8%8C%83)

作为接口路径，为了方便清晰的区分来自不同的系统，可以采用不同系统/模块名作为接口路径前缀，后面紧跟服务，最后为具体动作，如：问题模块的问题详情编辑：/issue/detail/update，基础模块产品信息添加：/system/product/add

## 版本控制规范[​](https://dotnetchina.gitee.io/furion/docs/specification-document" \l "633-%E7%89%88%E6%9C%AC%E6%8E%A7%E5%88%B6%E8%A7%84%E8%8C%83" \o "Direct link to heading)

为了便于后期接口的升级和维护，建议在接口路径中加入版本号，便于管理，实现接口多版本的可维护性。如：接口路径中添加类似"v1"、"v2"等版本号

## 接口命名规范[​](https://dotnetchina.gitee.io/furion/docs/specification-document" \l "634-%E6%8E%A5%E5%8F%A3%E5%91%BD%E5%90%8D%E8%A7%84%E8%8C%83" \o "Direct link to heading)

和 C# 命名规范一样，好的、统一的接口命名规范，不仅可以增强其可读性，而且还会减少很多不必要的口头/书面上的解释。可使用"驼峰命名法"按照实现接口的**业务类型、业务场景**等命名，有必要时可采取多级目录命名，但目录不宜过长，二级目录较为适宜

* 常见命名方式:
  + 接口名称动词前/后缀化： 接口名称以接口数据操作的动词为前/后缀，常见动词有：Add、Delete、Update、Query、Get、Send、Save、Detail、List、Page等，如：新建用户 AddUser、查询订单详情 QueryOrderDetail。
  + 接口名称动词 + 请求方式：接口路径中包含具体接口名称的名词，接口数据操作动作以 HTTP 请求方式来区分。常用的 HTTP 请求方式有：
    - GET：从服务器取出资源（一项或多项）
    - POST：在服务器新建一个资源
    - PUT：在服务器更新资源（客户端提供改变后的完整资源）
    - PATCH：在服务器更新资源（客户端提供改变的属性）
    - DELETE：从服务器删除资源
* 二级目录的使用方式，一级为服务、二级为动作，如下表。

|  |  |  |
| --- | --- | --- |
| 模块 | 一级命名 | 二级命名 |
| System | UserGroup | AddUserJoin |
|  |  | UserDetail |
|  | Product | Add |
|  |  | Page |

## 请求参数规范[​](https://dotnetchina.gitee.io/furion/docs/specification-document" \l "635-%E8%AF%B7%E6%B1%82%E5%8F%82%E6%95%B0%E8%A7%84%E8%8C%83" \o "Direct link to heading)

* 请求方式：按照 GET、POST、PUT 等含义定义，避免出现不一致现象，对人造成误解、歧义
  + 请求头：请求头根据项目需求添加配置参数。如：请求数据格式，accept=application/json 等。如有需要，请求头可根据项目需求要求传入用户 token、唯一验签码等加密数据
  + 请求参数/请求体： 请求参数字段，尽可能与数据库表字段、对象属性名等保持一致，因为保持一致是最省事，最舒服的一件事

## 返回数据规范[​](https://dotnetchina.gitee.io/furion/docs/specification-document" \l "636-%E8%BF%94%E5%9B%9E%E6%95%B0%E6%8D%AE%E8%A7%84%E8%8C%83" \o "Direct link to heading)

统一规范返回数据的格式，对己对彼都有好处，此处以 json 格式为例。返回数据应包含：**返回状态码、返回状态信息、具体数据**。**返回数据中的状态码、状态信息，常指具体的业务状态，不建议和 HTTP 状态码混在一起**。HTTP 状态，是用来体现 HTTP 链路状态情况，如：404-Not Found。HTTP 状态码和 json 结果中的状态码，并存尚可，用于体现不同维度的状态。

**{**

**"success": true,**

**"code": 0,**

**"message": "string",**

**"data": "string",**

**"extras": "string",**

**"timestamp": 0**

**}**

## 接口注解规范[​](https://dotnetchina.gitee.io/furion/docs/specification-document#636-%E8%BF%94%E5%9B%9E%E6%95%B0%E6%8D%AE%E8%A7%84%E8%8C%83)

接口的注解能够给开发人员熟悉接口，正确的使用输入、输出参数，同时自动生成swagger文档，便于同步到前端开发。

描述输入参数的每个字段的含义使用param name="input"标签。

描述输出参数的每个字段含义使用remarks标签。

**/// <summary>**

**/// 用户登录**

**/// </summary>**

**/// <param name="input">**

**/// Account：用户名 <br />**

**/// Password：密码**

**/// </param>**

**/// <remarks>用户登录接口备注说明<br />**

**/// 返回授权的accessToken </remarks>**

**/// <returns></returns>**

**public string LoginAsync([Required] LoginInput input)**

**/// <summary>**

**/// 用户登录**

**/// </summary>**

**/// <param name="input">**

**/// groupId：用户组ID**

**/// </param>**

**/// <remarks>备注说明<br />**

**/// 返回用户组<br />**

**/// GroupName：组名<br />**

**/// GroupID：组ID<br />**

**/// UserName：用户<br />**

**/// UserID：用户ID**

**/// </remarks>**

**/// <returns></returns>**

**public async Task<List<GroupUserOutput>> GetUserGroup(long groupId = 281695421571141)**

## 数据库命名

## 配置文件