API接口实现优化TODO

优化目标

根据各模块接口文档，确保后端实现与接口文档完全一致，遵循DDD分层架构规范。

通用优化指令

每个模块需执行的优化步骤

第1步：对比接口文档与现有实现

1. 阅读对应的接口文档（如 `03-用户管理模块接口文档.md`）
2. 查找对应的REST控制器（如 `UserRest.java`）
3. 对比接口端点、HTTP方法、请求参数、响应结构
4. 确认接口路径是否与文档一致（如 `/api/v1/users`）

第2步：HTTP方法和状态码规范

- [ ] **创建操作**：使用 `POST` + `@ResponseStatus(HttpStatus.CREATED)` 返回 201
- [ ] **全量更新**：使用 `PUT` + `@ResponseStatus(HttpStatus.OK)` 返回 200
- [ ] **部分更新/状态修改**：使用 `PATCH` + `@ResponseStatus(HttpStatus.OK)` 返回 200
- [ ] **删除操作**：使用 `DELETE` + `@ResponseStatus(HttpStatus.NO_CONTENT)` 返回 204，**无响应体**

```
// 创建 - POST 201
@ResponseStatus(HttpStatus.CREATED)
@PostMapping
public ApiLocaleResult<XxxDetailVo> create(@Valid @RequestBody
XxxCreateDto dto) {
    return ApiLocaleResult.success(xxxFacade.create(dto));
}

// 全量更新 - PUT 200
@ResponseStatus(HttpStatus.OK)
@PutMapping("/{id}")
public ApiLocaleResult<XxxDetailVo> update(@PathVariable Long id,
@Valid @RequestBody XxxUpdateDto dto) {
    return ApiLocaleResult.success(xxxFacade.update(id, dto));
}

// 状态修改 - PATCH 200
@ResponseStatus(HttpStatus.OK)
@PatchMapping("/{id}/status")
public ApiLocaleResult<XxxDetailVo> updateStatus(@PathVariable Long
id, @Valid @RequestBody XxxStatusDto dto) {
```

```
        return ApiLocaleResult.success(xxxFacade.updateStatus(id,
dto));
    }

    // 删除 - DELETE 204 无响应体
    @ResponseStatus(HttpStatus.NO_CONTENT)
    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) {
        xxxFacade.delete(id);
    }
```

第3步：Swagger注解规范

- [ ] 控制器类使用 `@Tag` 定义API分组
- [ ] 方法使用 `@Operation` 描述操作（含 `operationId`, `summary`, `description`）
- [ ] 使用 `@ApiResponses` 定义响应状态码说明
- [ ] 路径参数使用 `@Parameter` 描述
- [ ] 查询参数对象使用 `@ParameterObject` 注解

```
@Tag(name = "User", description = "用户管理 - 用户的创建、查询、修改、删
除")
@RestController
@RequestMapping("/api/v1/users")
public class UserRest {

    @Operation(operationId = "getUser", summary = "获取用户详情",
description = "根据ID获取用户详细信息")
    @ApiResponses(value = {
        @ApiResponse(responseCode = "200", description = "用户详情获
取成功"),
        @ApiResponse(responseCode = "404", description = "用户不存
在")
    })
    @GetMapping("/{id}")
    public ApiLocaleResult<UserDetailVo> getDetail(
        @Parameter(description = "用户ID") @PathVariable Long id) {
        return ApiLocaleResult.success(userFacade.getDetail(id));
    }

    @Operation(operationId = "listUsers", summary = "获取用户列表",
description = "分页获取用户列表")
    @GetMapping
    public ApiLocaleResult<PageResult<UserListVo>> list(
        @Valid @ParameterObject UserFindDto dto) {
        return ApiLocaleResult.success(userFacade.list(dto));
    }
}
```

第4步：DTO/VO参数检查

- ☐ 检查DTO参数是否与接口文档请求体一致
- ☐ 检查VO参数是否与接口文档响应体一致
- ☐ 补充缺失的DTO类（如状态更新、批量操作等）
- ☐ 补充缺失的VO类（如操作响应、统计信息等）
- ☐ 使用 `@Schema` 注解描述所有字段
- ☐ 使用Bean Validation注解进行参数校验，**不需要添加message属性**

```
// 正确示例
@NotBlank
@Size(max = 100)
@Schema(description = "名称", requiredMode = RequiredMode.REQUIRED)
private String name;

// 错误示例（不需要message）
@NotBlank(message = "名称不能为空")
private String name;
```

第5步：DTO继承规范

- ☐ **分页查询DTO** 必须继承 `PageQuery` 类，获得分页、排序、全文搜索能力，即内置分页和全文检索字段（pageNo、pageSize、infoScope、keyword、tenantId、createdBy、createdDate、modifiedBy、modifiedDate、filters）
- ☐ **创建/更新DTO** 直接定义，不需要继承

```
// 分页查询DTO - 继承PageQuery
@Data
@EqualsAndHashCode(callSuper = true)
@Schema(description = "查询用户请求参数")
public class UserFindDto extends PageQuery {
    @Schema(description = "用户名")
    private String username;

    @Schema(description = "状态")
    private UserStatus status;
}

// 创建DTO - 不继承
@Data
@Schema(description = "创建用户请求参数")
public class UserCreateDto {
    @NotBlank
    @Schema(description = "用户名", requiredMode =
RequiredMode.REQUIRED)
    private String username;
}
```

- ☐ **PageQuery定义**

```
public abstract class PageQuery {

  @Schema(description = "Field name to sort the data by")
  protected String orderBy;

  @Schema(description = "Specifies the direction of the sorting
(ascending or descending)")
  protected OrderSort orderSort;

  /**
   * @see SearchCriteria#INFO_SCOPE_KEY
   */
  @Schema(description = "Scope of information to query (BASIC or
DETAIL). "
      + "Interface performance optimization parameters, only valid for
some interfaces")
  public InfoScope infoScope;

  @JsonIgnore
  @Schema(description = "Whether to use full-text search (default:
false, uses DB index search if false)", hidden = true)
  public boolean fullTextSearch = false;

  @Schema(description = "Search keyword")
  private String keyword;

  @Schema(description = "Tenant ID to which this belongs")
  private Long tenantId;
  @Schema(description = "ID of the creator")
  private Long createdBy;
  @Schema(description = "Creation date, the format is `2024-10-12
00:00:00`")
  private LocalDateTime createdDate;
  @Schema(description = "ID of the last modifier")
  private Long modifiedBy;
  @Schema(description = "Last modification date, the format is `2024-10-
12 00:00:00`")
  private LocalDateTime modifiedDate;

  @Parameter(description = "Dynamic filter/search conditions (array of
SearchCriteria)",
      array = @ArraySchema(schema = @Schema(type = "object",
implementation = SearchCriteria.class)))
  protected List<SearchCriteria> filters = new ArrayList<>();
}
```

第6步：VO继承规范

- ☐ **详情VO** 继承 `TenantAuditingVo`，包含审计字段（tenantId, createdBy, creator, createdDate, modifiedBy, modifier, modifiedDate）
- ☐ **列表VO** 可继承详情VO或单独定义（通常字段更少）
- ☐ 使用 `@NameJoinField` 自动填充关联名称

```
// 详情VO - 继承TenantAuditingVo
@Data
@EqualsAndHashCode(callSuper = true)
@Schema(description = "用户详情")
public class UserDetailVo extends TenantAuditingVo {
    @Schema(description = "用户ID")
    private Long id;

    @Schema(description = "用户名")
    private String username;

    @Schema(description = "部门ID")
    private Long deptId;

    @Schema(description = "部门名称")
    @NameJoinField(id = "deptId", repository = "deptRepo")
    private String deptName;
}
```

第7步：日期字段规范

- ☐ 所有日期字段使用 `LocalDateTime` 类型
- ☐ 不需要手动格式化（框架自动处理）
- ☐ 检查VO中的日期字段类型

第8步：枚举类规范

- ☐ 检查接口文档中的枚举值
- ☐ 将枚举值定义为枚举类，**不需要考虑枚举属性字段和Message**
- ☐ 枚举类放到对应Entity所在目录（`domain/xxx/enums/`）

```
// 正确示例（简单枚举）
public enum UserStatus {
    ACTIVE,
    INACTIVE,
    PENDING,
    LOCKED
}

// 错误示例（不需要额外属性和message）
public enum UserStatus {
    ACTIVE("active", "活跃"),
    INACTIVE("inactive", "未激活");
```

```
    private final String code;
    private final String message;
}
```

第9步：门面层规范

- [ ] 门面层(Facade)注入 `XXXCmd`（命令服务）和 `XXXQuery`（查询服务）
- [ ] 门面层查询**必须**使用 `XXXQuery` 类方法，**禁止**直接使用 `XXXRepo`
- [ ] 使用 `Assembler` 进行 DTO → Domain → VO 转换
- [ ] 使用 `buildVoPageResult()` 构建分页结果
- [ ] 如Query接口缺少方法，需先在Query接口和实现类中添加

```
@Service
public class UserFacadeImpl implements UserFacade {
    @Resource
    private UserCmd userCmd;      // 命令服务（写操作）
    @Resource
    private UserQuery userQuery;  // 查询服务（读操作）

    @Override
    public UserDetailVo create(UserCreateDto dto) {
        // DTO -> Domain
        User user = UserAssembler.toCreateDomain(dto);
        // 调用命令服务
        User saved = userCmd.create(user);
        // Domain -> VO
        return UserAssembler.toDetailVo(saved);
    }

    @Override
    public PageResult<UserListVo> list(UserFindDto dto) {
        // Dto有name或者description字段，需要全文检索时
        GenericSpecification<User> spec =
UserAssembler.getSpecification(dto);
        Page<User> page = userQuery.find(spec, dto.tranPage(),
            dto.fullTextSearch,
getMatchSearchFields(dto.getClass()));

        // Dto没有name或者description字段，不需要全文检索时
        // GenericSpecification<User> spec =
UserAssembler.getSpecification(dto);
        // Page<User> page = userQuery.find(spec, dto.tranPage());
        return buildVoPageResult(page, UserAssembler::toListVo);
    }
}
```

第10步：应用层Cmd服务规范

- [ ] Cmd接口定义写操作方法，参数和返回值使用Domain对象
- [ ] Cmd实现类继承 `CommCmd<Entity, Long>` 获得基础CRUD能力
- [ ] 使用 `@Transactional` 保证事务性
- [ ] 使用 `BizTemplate` 进行业务处理（checkParams + process）

```
@Service
public class UserCmdImpl extends CommCmd<User, Long> implements
UserCmd {
    @Resource
    private UserRepo userRepo;
    @Resource
    private UserQuery userQuery;

    @Override
    @Transactional
    public User create(User user) {
        return new BizTemplate<User>() {
            @Override
            protected void checkParams() {
                if (userQuery.existsByUsername(user.getUsername()))
{
                    throw ResourceExisted.of("用户名已存在");
                }
            }
            @Override
            protected User process() {
                insert(user);
                return user;
            }
        }.execute();
    }

    @Override
    protected BaseRepository<User, Long> getRepository() {
        return userRepo;
    }
}
```

第11步：应用层Query服务规范

- [ ] Query接口定义读操作方法
- [ ] 使用 `BizTemplate` 进行业务处理
- [ ] 查询后批量设置关联数据（避免N+1问题）
- [ ] 支持全文搜索和标准查询两种模式

```
@Service
public class UserQueryImpl implements UserQuery {
    @Resource
```

```
    private UserRepo userRepo;
    @Resource
    private UserSearchRepo userSearchRepo;  // 全文搜索

    @Override
    public User findAndCheck(Long id) {
        return new BizTemplate<User>() {
            @Override
            protected User process() {
                return userRepo.findById(id)
                    .orElseThrow(() -> ResourceNotFound.of("用户不存
在"));
            }
        }.execute();
    }

    // 需要支持全文检索时
    @Override
    public Page<User> find(GenericSpecification<User> spec,
PageRequest pageable,
        boolean fullTextSearch, String[] match) {
        return new BizTemplate<Page<User>>() {
            @Override
            protected Page<User> process() {
              return fullTextSearch
                  ? userSearchRepo.find(spec.getCriteria(),
pageable, User.class, match)
                  : userRepo.findAll(spec, pageable);
            }
        }.execute();
    }

    // 不需要支持全文检索时
    @Override
    public Page<User> find(GenericSpecification<User> spec,
PageRequest pageable) {
        return new BizTemplate<Page<User>>() {
            @Override
            protected Page<User> process() {
              return userRepo.findAll(spec, pageable);
            }
        }.execute();
    }
}
```

第12步：Assembler组装器规范

- [ ] Assembler是静态工具类，提供静态方法进行转换
- [ ] `toCreateDomain()`: DTO → Domain（创建）
- [ ] `toUpdateDomain()`: DTO → Domain（更新）
- [ ] `toDetailVo()`: Domain → 详情VO

- ☐ `toListVo()`: Domain → 列表VO
- ☐ `getSpecification()`: DTO → 查询条件

```
public class UserAssembler {
    // DTO -> Domain（创建）
    public static User toCreateDomain(UserCreateDto dto) {
        User user = new User();
        user.setUsername(dto.getUsername());
        user.setEmail(dto.getEmail());
        return user;
    }

    // Domain -> VO
    public static UserDetailVo toDetailVo(User user) {
        UserDetailVo vo = new UserDetailVo();
        vo.setId(user.getId());
        vo.setUsername(user.getUsername());
        // 设置审计信息
        vo.setTenantId(member.getTenantId());
        vo.setCreatedBy(member.getCreatedBy());
        vo.setCreatedDate(member.getCreatedDate());
        vo.setModifiedBy(member.getModifiedBy());
        vo.setModifiedDate(member.getModifiedDate());
        return vo;
    }

    // DTO -> 查询条件
    public static GenericSpecification<User>
getSpecification(UserFindDto dto) {
        Set<SearchCriteria> filters = new SearchCriteriaBuilder<>
(dto)
            .rangeSearchFields("id", "createdDate")
            .orderByFields("id", "createdDate", "username")
            .matchSearchFields("username", "email") // 配置全文检索字
段
            .build();
        return new GenericSpecification<>(filters);
    }
}
```

第13步：方法顺序规范

接口层、门面层、业务层统一按以下顺序排列方法：

1. **创建**（create, add, insert）
2. **更新**（update, modify, edit）
3. **修改状态**（enable/disable, lock/unlock, activate, resetPassword等）
4. **删除**（delete, remove, batchDelete）
5. **查询详细**（getDetail, findById, get）

6. **查询列表**（list, find, search）
7. **查询统计**（getStats, count, statistics）

> **注意**：其他业务操作放在统计之后，按业务逻辑分组

第14步：代码注释规范

- [ ] 不使用分组注释分隔不同类型的方法（不需要下面注解）

```
// ==================== 创建 ====================
// ==================== 更新 ====================
// 不需要这些分组注释
```

第15步：JPA JSON字段规范

- [ ] JSON字段使用对象类型，不要手动编写序列化
- [ ] 使用 `@Type(JsonType.class)` 注解

```
// 正确示例
@Type(JsonType.class)
@Column(name = "tags", columnDefinition = "json")
private List<String> tags;

@Type(JsonType.class)
@Column(name = "config", columnDefinition = "json")
private ConfigObject config;  // 使用对象类型

// 错误示例（不要手动序列化）
@Column(name = "config")
private String configJson;  // 然后手动JSON.parse/stringify
```

第16步：异常处理规范

- [ ] 使用框架提供的统一异常类
- [ ] 资源不存在：`ResourceNotFound.of()`
- [ ] 资源已存在：`ResourceExisted.of()`
- [ ] 协议异常：`ProtocolException.of()`

```
// 资源不存在
throw ResourceNotFound.of("用户不存在");

// 资源已存在
throw ResourceExisted.of("用户名已存在");
```

第17步：分页响应规范

- ☐ 分页查询返回 `PageResult<T>` 类型
- ☐ 使用 `buildVoPageResult()` 构建分页结果
- ☐ 分页字段：`total`（总数）、`list`（数据列表）

```
// Facade层构建分页结果
return buildVoPageResult(page, UserAssembler::toListVo);
```

模块处理清单

已完成

- ☑ **01-认证授权模块** - AuthRest.java ✅
- ☑ **02-租户管理模块** - TenantRest.java ✅
- ☑ **03-用户管理模块** - UserRest.java ✅
- ☑ **04-部门管理模块** - DepartmentRest.java ✅
- ☑ **05-组管理模块** - GroupRest.java ✅
- ☑ **06-应用管理模块** - ApplicationRest.java ✅
- ☑ **07-服务管理模块** - ServiceRest.java ✅
- ☑ **08-接口管理模块** - InterfaceRest.java ✅
- ☑ **09-标签管理模块** - TagRest.java ✅
- ☑ **10-权限策略模块** - PolicyRest.java ✅
- ☑ **11-授权管理模块** - AuthorizationRest.java ✅
- ☑ **12-消息通知模块** - NotificationRest.java ✅ （新建，路径/api/v1/notifications）
- ☑ **13-备份恢复模块** - BackupRest.java ✅
- ☑ **14-短信消息模块** - SmsRest.java ✅
- ☑ **15-电子邮件模块** - EmailRest.java ✅
- ☑ **16-安全设置模块** - SecurityRest.java ✅
- ☑ **17-系统监控模块** - MonitoringRest.java ✅
- ☑ **18-接口监控模块** - ApiMonitoringRest.java ✅ （路径/api/v1/interface-monitoring）
- ☑ **19-LDAP集成模块** - LdapRest.java ✅
- ☑ **20-资源配额模块** - QuotaRest.java ✅ （路径/api/v1/quotas）
- ☑ **21-审计日志模块** - AuditLogRest.java ✅ （路径/api/v1/audit-logs）
- ☑ **22-系统版本模块** - SystemVersionRest.java ✅ （路径/api/v1/system-version）

待处理

无

DDD分层结构参考

```
core/src/main/java/cloud/xcan/angus/core/gm/
│
├── interfaces/           # 接口层
│   └── {module}/
│       ├── {Module}Rest.java              # REST控制器
│       └── facade/
│           ├── {Module}Facade.java        # 门面接口
│           ├── dto/                       # 请求DTO
│           │   ├── {Module}CreateDto.java
│           │   ├── {Module}UpdateDto.java
│           │   ├── {Module}FindDto.java
│           │   └── {Module}XXXDto.java    # 其他操作DTO
│           ├── vo/                        # 响应VO
│           │   ├── {Module}DetailVo.java
│           │   ├── {Module}ListVo.java
│           │   ├── {Module}StatsVo.java
│           │   └── {Module}XXXVo.java     # 其他响应VO
│           └── internal/
│               ├── {Module}FacadeImpl.java
│               └── assembler/
│                   └── {Module}Assembler.java
│
├── application/          # 应用层
│   ├── cmd/
│   │   └── {module}/
│   │       ├── {Module}Cmd.java           # 命令接口
│   │       └── impl/
│   │           └── {Module}CmdImpl.java   # 命令实现
│   └── query/
│       └── {module}/
│           ├── {Module}Query.java         # 查询接口
│           └── impl/
│               └── {Module}QueryImpl.java # 查询实现
│
├── domain/               # 领域层
│   └── {module}/
│       ├── {Module}.java                  # 领域实体
│       ├── {Module}Repo.java              # 仓储接口
│       └── enums/                         # 枚举目录
│           └── {Enum}Status.java          # 状态枚举
│
└── infra/                # 基础设施层
    └── persistence/
        └── mysql/
            └── {module}/
                └── {Module}RepoMysql.java
```

使用说明

处理单个模块

```
请根据 {XX-模块名接口文档.md} 和 后端开发接口说明.md,
按照上述优化指令，修改 {ModuleRest.java} 接口实现。
```

示例指令

```
请根据 04-部门管理模块接口文档.md 和 后端开发接口说明.md,
对比现有的DepartmentRest.java实现，按照以下要求进行优化：
1. 检查DTO和VO参数是否有遗漏
2. 日期字段使用LocalDateTime
3. 枚举值定义成枚举类（不需要考虑枚举属性字段和Message）
4. 门面层查询使用Query类方法
5. 方法顺序调整为：创建、更新、修改状态、删除、查询详细、查询列表、查询统计
6. JPA JSON字段使用对象类型，不要手动序列化
7. DTO校验注解不需要添加message属性
```

注意事项

1. **保持一致性**: 所有模块遵循相同的规范和结构
2. **先查后改**: 修改前先充分了解现有实现
3. **逐层修改**: 按 REST → Facade → Cmd/Query 顺序修改
4. **编译验证**: 每次修改后确保编译通过
5. **保留TODO**: 未实现的业务逻辑用TODO标记

优化要求摘要

序号	优化项	说明
1	HTTP方法和状态码	POST→201, PUT/PATCH→200, DELETE→204无响应体
2	Swagger注解	@Tag, @Operation, @ApiResponses, @Parameter, @ParameterObject
3	DTO/VO参数检查	与接口文档保持一致
4	DTO继承规范	查询DTO继承PageQuery，创建/更新DTO不继承
5	VO继承规范	详情VO继承TenantAuditingVo，使用@NameJoinField填充关联名称
6	日期字段	使用LocalDateTime，不手动格式化
7	枚举类	简单枚举，不需要属性字段和Message
8	门面层规范	注入Cmd+Query，使用Assembler转换，buildVoPageResult分页
9	Cmd服务规范	继承CommCmd，@Transactional事务，BizTemplate业务处理

序号	优化项	说明
10	Query服务规范	支持全文搜索，批量设置关联数据避免N+1
11	Assembler规范	toCreateDomain, toDetailVo, getSpecification
12	方法顺序	创建→更新→修改状态→删除→查询详细→查询列表→查询统计
13	JPA JSON字段	使用对象类型 + @Type(JsonType.class)
14	DTO校验注解	不需要添加message属性
15	异常处理	使用ResourceNotFound, ResourceExisted等统一异常类
16	分页响应	使用PageResult，buildVoPageResult构建