## description: 通用编码规范与最佳实践，适用于项目所有文件。 globs: alwaysApply: true

# 全局代码规范与最佳实践

## Critical Rules

### 核心原则

* **奥卡姆剃刀原则 (Occam's Razor):** 始终选择最简单有效方案。如无绝对必要，勿增实体（数据库表、UI元素、代码模块、测试用例等）。保持设计和实现的简洁性。
* **AI 交互语言:** AI 助手必须使用中文进行回答和沟通。
* **MDC 文件语言:** 项目中的 .mdc 规则文件必须使用中文编写。
* **MDC 文件位置:**.mdc 文件必须放置于项目根目录下的 .cursor/rules/ 文件夹内。
* **分步代码修改:** AI 助手在修改代码时，必须遵循分步操作原则，一次只完成一个明确、独立的修改任务。

### 开发实践

* **DRY (Don't Repeat Yourself):** 严格避免代码重复。在编写或修改代码前，务必检查代码库中是否存在可复用的相似功能或逻辑。优先抽取和复用公共组件、函数或服务。
* **目标驱动修改:** 仅针对明确的需求或已充分理解的任务进行代码更改。修改范围应严格控制在相关模块内，避免无目的或超出范围的修改。
* **稳定性优先 (修复 Bug 时):** 修复问题时，优先在现有技术和架构内解决。避免引入新的技术、库或设计模式，除非现有方案确实无法解决且新方案已充分评估。若引入新方案，必须彻底移除被替代的旧逻辑。
* **代码整洁:** 保持代码库的整洁和有序。及时移除无用代码、注释掉的代码块和不再使用的文件。
* **脚本管理:** 避免在项目源代码文件中直接编写一次性脚本（例如数据迁移脚本）。应将此类脚本放在独立的、指定的目录中管理。
* **模块化与封装:** 当函数或文件代码行数超过 300-500 行时，应考虑将其重构，拆分成更小、更专注的函数或模块。
* **配置文件安全:** 禁止直接覆盖 .env 或其他核心配置文件。任何修改必须经过确认和（若有必要）团队同意。

### uniapp最佳实践

* **Vue.js 版本选择:** 默认使用 Vue 3 进行项目开发。Vue 3 提供了更好的性能、Composition API 和更现代的开发体验。
* **tailwind 纯css版引入:** 使用 static\css\tailwind.css 文件，已经针对微信小程序进行了优化，可直接使用 tailwind css 语法，不会有语法错误。如果本地没有该文件，必须从 https://cdn.bitejufeng.com/dev/tailwind.css 下载到本地使用。
* **font Awesome 引入:** 使用 static\css\awesome-font.css 文件，采用纯 CSS 调用方式，无需使用组件。如果本地没有该文件，必须从 https://cdn.bitejufeng.com/dev/awesome-font.css 下载到本地使用。
* **云开发平台:** 项目默认采用 uniCloud 作为云开发平台,uniCloud默认使用阿里云
* **文档提示:** 开发过程中，应使用context7 mcp查阅并遵循 uni-app 及 uniCloud 官方文档
* **Schema先行:** 云数据库设计应采用 DB Schema 定义，结合索引优化，并利用 schema2code 提升开发效率。 生成DB Schema之后,要生成配套的 表名.init\_data.json：数据表初始化数据 表名.index.json：表的索引配置
* **云开发文件夹**云开发操作文件夹为uniCloud-aliyun(必定是这个文件夹下创建db schema)
* **规范命名:** 云数据库表和云对象必须遵循统一的命名规则（必须是 a-{模块名}）以增强可维护性。
* **分层操作数据库:**
  + **JQL 语法统一:** ClientDB 和云对象都默认使用 JQL 语法进行数据库操作，确保语法一致性和开发体验统一。
  + **ClientDB 使用场景：** 用于简单的业务数据查询和操作（如 banner 列表、商品列表、订单查询、用户个人信息管理等）。权限微调后的 uni-id-users 表也可以使用 ClientDB 进行简单的 CRUD 操作。
  + **云对象使用场景：** 复杂业务逻辑、多表关联计算、第三方API调用、敏感权限操作、支付处理、批量事务操作。当 ClientDB 无法满足复杂业务需求时使用云对象实现。
* **云对象封装业务:** 核心业务逻辑和复杂数据处理应通过云对象实现。
* **云对象方法调用限制:** 云对象导出的不同方法之间不能互相调用。必须将共享逻辑抽取到模块外部的独立函数中供各方法调用。
* **标准化用户认证:** 用户管理和认证应基于 uni-id，优先采用微信一键手机号登录
* **统一配置管理:** 利用 uni-config-center 集中管理项目配置。配置文件在uni\_modules\uni-config-center\uniCloud\cloudfunctions\common
* **模块化依赖:** 云对象依赖（如 uni-id-common）应在 package.json 中明确声明。
* **善用云存储:** 文件（图片、音视频等）存储首选 uniCloud 云存储。
* **组件化前端:** 前端 UI 构建推荐使用成熟的组件库（如 tailwind,awesome-font）以加速开发。
* **关键日志:** 在数据请求、云函数调用等关键节点添加必要的日志记录，便于调试和追踪问题。
* **接口参数对象化:** 前端调用后端接口时，推荐使用对象形式传递参数，增强可读性和可扩展性。
* **云对象测试:** 为云对象编写测试用例（如使用 .param.js 文件），确保逻辑正确性。
* **运行时版本统一:** 云对象推荐使用统一且较新的运行环境（如 Node.js 16）

### 云对象方法调用规范 ⚠️

**重要限制**：云对象导出的不同方法之间**不能**互相调用。这是 uniCloud 云对象的重要限制。

#### 错误示例（会导致运行时错误）

// ❌ 错误：云对象方法间无法互相调用module.exports = { asynctryAddTodo(title, content) { try { returnthis.addTodo(title, content); // ❌ 错误！this.addTodo is not a function } catch (e) { return { errCode: 'add-todo-failed' } } }, asyncaddTodo(title, content) { // 添加 todo 的逻辑return { success: true } } }

#### 正确示例（抽取公共函数）

// ✅ 正确：将共享逻辑抽取到模块外部asyncfunctionpureAddTodo(title, content) { // 添加 todo 的具体逻辑const db = uniCloud.databaseForJQL({ clientInfo: this.getClientInfo() }); const result = await db.collection('todos').add({ title, content, created\_at: newDate() }); return result; } module.exports = { asynctryAddTodo(title, content) { try { returnawait pureAddTodo.call(this, title, content); // ✅ 正确调用 } catch (e) { return { errCode: 'add-todo-failed', errMsg: e.message } } }, asyncaddTodo(title, content) { returnawait pureAddTodo.call(this, title, content); // ✅ 正确调用 } }

#### 规范要点

1. **公共函数位置**：将共享逻辑定义在 module.exports 之前的独立函数中
2. **this 上下文传递**：使用 .call(this, ...) 或 .apply(this, ...) 传递云对象的上下文
3. **async/await 处理**：公共函数的 async 特性根据业务需要决定，不是必须的
4. **错误处理**：在公共函数中进行统一的错误处理和日志记录
5. **命名规范**：公共函数建议以 pure 或 \_ 前缀命名，表明其为内部工具函数

#### 实际应用示例

// 会员服务云对象的正确写法const uniIdCommon = require('uni-id-common'); // 公共函数：查询用户会员信息asyncfunctionqueryMemberByUserId(userId) { const db = uniCloud.databaseForJQL({ clientInfo: this.getClientInfo() }); const result = await db.collection('a-members') .where(`user\_id == "${userId}"`) .get(); return result.data.length > 0 ? result.data[0] : null; } // 公共函数：初始化用户会员asyncfunctioninitMemberRecord(userId) { const db = uniCloud.databaseForJQL({ clientInfo: this.getClientInfo() }); const memberData = { user\_id: userId, level: 'normal', status: 'active' }; await db.collection('a-members').add(memberData); return memberData; } module.exports = { \_before() { this.uniIdCommon = uniIdCommon.createInstance({ context: this }); }, asyncgetUserMemberInfo() { const checkResult = awaitthis.uniIdCommon.checkToken(this.getUniIdToken()); if (checkResult.errCode !== 0) { thrownewError('身份验证失败'); } // 调用公共函数let memberInfo = await queryMemberByUserId.call(this, checkResult.uid); if (!memberInfo) { memberInfo = await initMemberRecord.call(this, checkResult.uid); } return { errCode: 0, data: memberInfo }; }, asyncupdateMemberLevel(newLevel) { // 也可以复用相同的公共函数const memberInfo = await queryMemberByUserId.call(this, this.getCurrentUserId()); // 更新逻辑... } }

### uni\_modules 插件机制

* **插件云函数配置:** uni\_modules 插件中常常会需要有对应的云函数和配置类。
* **自动映射机制:** uni\_modules 中的云函数部分，通常会被编译器自动映射到 uniCloud-aliyun\cloudfunctions 文件夹下。
* **文件缺失正常现象:** 如果在 uniCloud-aliyun\cloudfunctions 看到文件缺失，是很正常的。实际上已经在 uni\_modules 中被配置了。
* **重要配置模块:**uni-config-center 也是一个 uni\_modules 插件，用于统一配置管理。
* **配置位置:** 配置的位置在 uni\_modules\uni-config-center\uniCloud\cloudfunctions\common 文件夹中。
* **uni-id 配置集成:** 其中有 uni-id 配置的处理。uni-id 这个配置在 uni-module 中已添加，会被自动映射到正确的文件夹目录下。
* **uni-id 表 Schema 位置:** uni-id 相关的表 schema 文件位于 uni\_modules\uni-id-pages\uniCloud\database 路径下，包括 uni-id-users.schema.json、uni-id-roles.schema.json、uni-id-permissions.schema.json 等。不要在 uniCloud-aliyun\database 下重复创建这些文件。

### 云对象中的 uni-id Token 处理规范

* **标准依赖引入:** 云对象中必须使用 const uniIdCommon = require('uni-id-common') 引入 uni-id 公共模块。
* **实例初始化:** 在 \_before 方法中创建 uni-id 实例：this.uniIdCommon = uniIdCommon.createInstance({ context: this })。
* **Token 验证方法:** 使用 await this.uniIdCommon.checkToken(this.getUniIdToken()) 方法验证并解析用户 token。
* **安全验证原则:** 所有涉及用户身份的操作都必须通过 checkToken 方法验证，禁止直接使用前端传递的用户ID参数。
* **错误处理:** 必须检查 checkTokenResult.errCode 是否为 0，非零时抛出相应错误信息。
* **用户信息获取:** 从 checkTokenResult 中获取 uid（用户ID）、role（角色）、permission（权限）等信息。
* **Token 自动刷新:** 利用 checkTokenResult.token 和 checkTokenResult.tokenExpired 实现 token 自动刷新机制。
* **依赖配置:** 云对象的 package.json 中必须声明 uni-id-common 依赖。
* **禁止手动解析:** 严禁手动解析 JWT token 字符串，必须使用官方提供的 checkToken 方法。

### JQL 语法使用规范（ClientDB & 云对象）

* **JQL 语法统一:** ClientDB 和云对象都默认使用 JQL 语法，提供一致的数据库操作体验。
* **前端 ClientDB JQL 写法:**// 前端 ClientDB 使用 JQLconst db = uniCloud.database(); const result = await db.collection('table\_name').where('field == value').field('field1, field2').get();
* **云对象 JQL 数据库引用:** 在云对象中使用 JQL 语法时，必须使用 uniCloud.databaseForJQL({ clientInfo: this.getClientInfo() }) 获取数据库引用，传入客户端信息以确保权限校验正常工作。
* **云对象 JQL 标准写法:**// 云对象中使用 JQLconst db = uniCloud.databaseForJQL({ clientInfo: this.getClientInfo() }); const result = await db.collection('table\_name').where('field == value').get();
* **权限校验保障:** 传入 clientInfo 确保 JQL 查询能够正确执行 Schema 中定义的权限校验逻辑。
* **字段选择优化:** 优先使用 JQL 的字符串格式字段选择：.field('field1, field2, field3') 而非对象格式。

### 用户信息操作规范

* **uni-id-users 权限微调:**uni\_modules\uni-id-pages\uniCloud\database\uni-id-users.schema.json 的权限设置可以按照具体业务需求进行微调。通过修改字段的 permission.write 属性，可以让用户拥有修改自己信息的权限，格式为："write": "doc.\_id == auth.uid || 'CREATE\_UNI\_ID\_USERS' in auth.permission || 'UPDATE\_UNI\_ID\_USERS' in auth.permission"。
* **ClientDB 优先策略:** 权限微调后，优先使用 ClientDB 实现简单的用户信息增删改查操作。ClientDB 会自动处理权限校验，代码更简洁，开发效率更高。
* **权限微调范围:** 常见的可微调字段包括：username（用户名）、nickname（昵称）、avatar（头像）、gender（性别）、mobile（手机号）、email（邮箱）、birthday（生日）、realname\_auth（实名认证信息）等用户个人可管理的信息。
* **云对象备用方案:** 仅在 ClientDB 无法满足复杂业务逻辑需求时（如多表关联、复杂数据验证、第三方API调用等），才使用云对象实现。
* **数据安全保障:** 无论使用 ClientDB 还是云对象，都必须确保用户只能操作自己的信息，避免越权操作。
* **业务逻辑分层:** 简单的 CRUD 操作使用 ClientDB，复杂的业务逻辑（如积分计算、会员等级升级、数据统计等）使用云对象实现。

### pages.json 配置规范

* **页面路由配置:** 所有页面必须在 pages.json 中正确配置路由路径，遵循 uni-app 路由规范。
* **uniIdRouter 集成:** 项目必须集成 uniIdRouter 实现页面访问控制，确保用户认证状态管理的统一性。
* **登录拦截配置:** 在 pages.json 的 uniIdRouter 配置中，明确指定需要登录才能访问的页面列表。
* **自动跳转机制:** 未登录用户访问受保护页面时，系统应自动跳转到登录页面，登录成功后自动返回原页面。
* **页面访问权限分级:**
  + **公开页面:** 首页、商品列表、商品详情等无需登录即可访问
  + **登录页面:** 个人中心、订单管理、收藏列表、出行人管理等需要登录
  + **会员页面:** VIP专享内容、高级功能等需要特定会员等级
* **uniIdRouter 标准配置示例:**{"uniIdRouter":{"loginPage":"/pages/login/login","needLogin":["/pages/profile/profile","/pages/order/order-list","/pages/favorites-list/favorites-list","/pages/travelers/travelers"],"resToLogin":true}}
* **路由守卫最佳实践:** 结合 uni-id 的登录状态检查，在页面 onLoad 生命周期中进行二次验证，确保安全性。
* **页面级登录状态检查示例:**// 在需要登录的页面中添加登录状态检查exportdefault { asynconLoad() { // 检查用户登录状态const token = uni.getStorageSync('uni\_id\_token') if (!token) { // 未登录，跳转到登录页面 uni.navigateTo({ url: '/pages/login/login' }) return } // 验证token有效性（可选）try { const result = await uniCloud.callFunction({ name: 'uni-id-co', data: { action: 'checkToken' } }) if (result.result.code !== 0) { // token无效，清除本地存储并跳转登录 uni.removeStorageSync('uni\_id\_token') uni.navigateTo({ url: '/pages/login/login' }) return } } catch (error) { console.error('验证登录状态失败:', error) } } }
* **登录成功后的跳转处理:** 在登录页面实现登录成功后的智能跳转逻辑，支持返回原页面或默认首页。
* **页面needLogin属性:** 在 pages.json 中为每个页面配置 needLogin 属性，明确标识该页面的访问权限要求。