**约定规范文档**

**游乐园智慧助手**

作 者：姜堂蕴之 孟祥硕 陈金铭 池跃花

创建日期：2024年4月16日

版 本：v3.0

文档控制

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **变更内容** | **责任人** | **日期** |
| 1 | 初稿 | 姜堂蕴之 孟祥硕 | 2024-04-16 |
| 2 | 添加部分规约及示例 | 孟祥硕 | 2024-04-17 |
| 3 | 添加前端规范 | 陈金铭 池跃花 | 2024-04-17 |

**目录**

**[前端规范 4](#_Toc13229)**

[一、 编程规约 4](#_Toc29583)

[二、 Vue 项目规范 5](#_Toc23368)

**[后端规范 6](#_Toc19819)**

[一、 编程规约 6](#_Toc23757)

[二、 安全规约 10](#_Toc6200)

[三、 数据库规约 11](#_Toc6256)

[四、 工程结构 11](#_Toc29386)

# **前端规范**

1. **编程规约**
2. **命名规范**
3. 目录命名

全部采用小写方式，有复数结构时，要采用复数命名法， 缩写不用复数。

示例：page/home

1. 图片文件命名

全部采用小写方式， 以下划线分隔。

示例：map/cater\_position.png

1. 命名严谨性

代码中的命名严禁使用拼音与英文混合的方式，更不允许直接使用中文的方式。 说明：正确的 英文拼写和语法可以让阅读者易于理解，避免歧义。注意，即使纯拼音命名方式也要避免采用。

1. **Vue Template规范**
2. 缩进

缩进使用 4 个空格（一个 tab），嵌套的节点应该缩进。

1. 分块注释

在每一个块状元素，列表元素和表格元素前，加上一对 HTML 注释。

注释格式：<!-- 这是一个注释 -->

1. 引号

类名使用双引号(“”) 而不是单引号(‘’) 。

示例：<view class=”box”></view>

1. **CSS 规范**
2. 命名

类名使用小写字母，以下划线分隔。名称能反映元素的目的和用途。

示例：.important\_image{ }

1. 尽量使用缩写属性

示例：padding: 10px 20px;

1. 省略0后面的单位

示例：padding-bottom: 0;

1. 每个选择器及属性独占一行
2. 避免使用 ID 选择器及全局标签选择器防止污染全局样式
3. **Javascript 规范**
4. 命名

方法名、参数名、成员变量、局部变量都统一使用 lowerCamelCase风格，必须遵从驼峰形式。

示例：getHttpMessage()

1. 代码格式

使用 4 个空格进行缩进。

不同逻辑、不同语义、不同业务的代码之间插入一个空行分隔开来以提升可读性。

1. 条件判断和循环最多三层

超过三层抽象成函数。

1. this 的转换命名

对上下文this的引用只能使用‘self’来命名。

1. **Vue 项目规范**
2. 标签顺序保持一致

应该总是让标签顺序保持为:

<template>...</template>

<script>...</script>

<style>...</style>

1. 必须为v-for设置键值key
2. v-show 与 v-if 选择

如果运行时，需要非常频繁地切换，使用 v-show ；如果在运行时，条件很少改变，使用 v-if。

1. script 标签内部结构顺序

components > props > data > computed > watch > filter > 钩子函数（钩子函数按其执行顺序） > methods

1. 下列关键字后必须有大括号（即使代码块的内容只有一行）：if, else, for, while, do, switch, try, catch, finally, with。

# **后端规范**

1. **编程规约**
2. **命名风格**

在编程实践中，命名规范是确保代码清晰、易于理解的关键要素。以下是我们团队在命名方面所遵循的规约：

1. **开头与结尾字符**

代码中所有命名应避免以下划线（\_）或美元符号（$）作为开头或结尾字符，以确保命名的规范性和一致性。

示例：禁止使用\_name / \_\_name / $name / name\_ / name$ / name\_\_

1. **语言选择**

所有编程相关的命名应严格避免使用拼音与英文混合的方式，更不得直接使用中文命名。这一规约有助于保持代码的国际化，方便团队成员之间的协作与交流。

示例：taobao / cainiao/ aliyun/ youku / hangzhou

1. **类名命名**

类名应采用UpperCamelCase风格，即每个单词的首字母大写，其余字母小写。然而，对于特定类型的类名，如数据对象（DO）、业务对象（BO）、数据传输对象（DTO）、视图对象（VO）以及应用对象（AO）等，可以采用其他约定俗成的命名方式。

示例：ForceCode / UserDO / HtmlDTO / XmlService / TcpUdpDeal / TaPromotion

1. **方法、参数与变量命名**

方法名、参数名、成员变量以及局部变量应统一采用lowerCamelCase风格，即首个单词首字母小写，后续单词首字母大写。这种命名方式有助于提高代码的可读性。

示例：localValue / getHttpMessage() / inputUserId

1. **常量命名**

常量命名应全部采用大写字母，单词之间用下划线（\_）隔开。这种命名方式有助于快速识别常量，并避免与其他类型的变量混淆。

示例：MAX\_STOCK\_COUNT / CACHE\_EXPIRED\_TIME

1. 杜绝完全不规范的缩写，避免望文不知义
2. 避免在子父类的成员变量之间、或者不同代码块的局部变量之间采用完全相同的命名， 使可理解性降低
3. **包名命名**

包名应统一使用小写字母，并使用点（.）作为分隔符。每个分隔符之间应包含一个具有自然语义的英语单词。此外，包名应采用单数形式，除非类名具有复数含义，此时类名可以使用复数形式。

示例：com.chinasoft.backend

1. **避免重复命名** 在编写代码时，应尽量避免在子父类的成员变量之间或不同代码块的局部变量之间使用完全相同的命名，以免降低代码的可理解性。通过采用具有区分度的命名方式，可以提高代码的可读性和可维护性。
2. 在常量与变量的命名时，表示类型的名词放在词尾，以提升辨识度  
    示例：startTime / workQueue / nameList
3. **接口和实现类的命名** 对于Service和DAO类，基于SOA的理念，暴露出来的服务一定是接口，内部的实现类用 Impl 的后缀与接口区别  
    示例：CacheServiceImpl 实现 CacheService 接口
4. 枚举类名带上Enum后缀，枚举成员名称需要全大写，单词间用下划线隔开
5. Service/DAO 层方法命名规约   
    1） 获取单个对象的方法用get做前缀。   
    2） 获取多个对象的方法用list做前缀，复数结尾，如：listObjects。   
    3） 获取统计值的方法用count做前缀。   
    4） 插入的方法用save/insert做前缀。   
    5） 删除的方法用remove/delete做前缀。   
    6） 修改的方法用update做前缀。
6. 不允许任何魔法值（即未经预先定义的常量）直接出现在代码中。
7. 在long或者Long赋值时，数值后使用大写字母L，不能是小写字母l，小写容易跟 数字混淆，造成误解。
8. 不要使用一个常量类维护所有常量，要按常量功能进行归类，分开维护。
9. **代码格式**
10. **大括号使用约定**
11. 空代码块：当大括号内无代码时，应简洁地写为{}，无需换行。
12. 非空代码块：

* 左大括号{前不应换行。
* 左大括号{后应换行。
* 右大括号}前应换行。
* 若右大括号}后跟随else等代码，则不换行；若表示代码块终止，则必须换行。

1. **空格使用约定**
2. 左小括号(与字符之间，以及右小括号)与字符之间，均不应出现空格。
3. if、for、while、switch、do等保留字与紧随其后的括号之间，必须加一个空格。
4. 任何二目、三目运算符的左右两边，均应加一个空格，以提高代码可读性。
5. 注释的双斜线//与注释内容之间，应有一个空格，以保持注释的整洁性。
6. 在定义和传入方法参数时，多个参数之间的逗号后必须加一个空格，以提高代码的可读性。

|  |
| --- |
| public static void main(String[] agrs){  // 缩进4个空格  String say = "hello";  // 运算符的左右必须有一个空格  int flag = 0;  // 关键词 if 与括号之间必须有一个空格，括号内的 f 与左括号，0 与右括号不需要空格  if (flag == 0) {  System.out.println(say);  }    // 左大括号前加空格且不换行，左大括号后换行  if (flag == 1) {  System.out.println("world");  } else {  System.out.println("ok");  // 在有右大括号后直接结束，则必须换行  }  } |

1. **缩进与Tab字符** 采用4个空格作为缩进标准，禁止使用Tab字符，以确保代码在不同编辑器中的一致性。
2. **注释的双斜线与注释内容之间有且仅有一个空格** 示例：// 这是示例注释，请注意在双斜线之后有一个空格   
    String commentString = new String();
3. **换行原则**

单行字符数不得超过120个，超出时需进行换行。换行时应遵循以下原则：

1. 第二行相对于第一行缩进4个空格，从第三行开始不再继续缩进。
2. 运算符应与下文一同换行。
3. 方法调用的点符号应与下文一同换行。
4. 方法调用时，若多个参数需换行，应在逗号后进行。
5. 不应在括号前换行。
6. **控制语句**
7. 在一个 switch 块内部，每个 case 分支应通过 break、return 等语句来明确终止。在一个完整的 switch 块中，需要包含一个 default 语句，并将其放置在所有 case 分支之后。
8. 在编写 if/else/for/while/do 语句时，无论语句体内部只有一行代码还是多行代码，都应使用大括号来明确界定语句块的范围。
9. **代码组织**
10. 方法体内的执行语句组、变量的定义语句组、不同的业务逻辑或语义之间，应插入一个空行，以便于代码的阅读与维护。
11. 相同业务逻辑和语义之间的代码，则无需插入空行，以保持代码的紧凑性。
12. **OOP规约**
13. **静态成员访问**

避免通过对象引用访问静态成员：在访问类的静态变量或静态方法时，应直接使用类名而非通过对象引用。这样做可以避免无谓地增加编译器的解析成本，提高代码的执行效率。

1. **方法覆写注解**

了确保代码的可读性和维护性，所有覆写父类的方法都应明确标注@Override注解。这有助于编译器在编译时检查方法签名的正确性，并在重构过程中提供更强的类型检查。

1. **包装类对象比较**

包装类对象比较应使用equals方法：对于所有相同类型的包装类对象，比较其值时应统一使用equals方法，而非直接比较引用。这样可以确保比较的是对象的实际值，而非其在内存中的地址，从而避免潜在的比较错误。  
 Object的equals方法容易抛空指针异常，应使用常量或确定有值的对象来调用equals。  
 示例："test".equals(object);

1. **final关键字的使用**

合理使用final关键字：final关键字可用于声明类、成员变量、方法以及本地变量，以表示其不可变性。在以下情况下应使用final关键字：

1. 当一个类不允许被继承时，如String类，应将其声明为final类。
2. 当一个成员变量的引用不允许被修改时，如POJO类的域变量，应将其声明为final变量。
3. 当一个方法不允许被子类重写时，如POJO类的setter方法，应将其声明为final方法。
4. 当一个局部变量在运行过程中不允许重新赋值时，应使用final修饰。
5. 当需要避免上下文重复使用一个变量时，使用final描述可以强制重新定义一个变量，有助于更好地进行代码重构。
6. 定义数据对象DO类时，属性类型要与数据库字段类型相匹配  
    示例：数据库字段的bigint必须与类属性的Long类型相对应。
7. **注释规约**
8. **Javadoc 规范注释**
9. 类、类属性、类方法注释：为确保代码的可读性和可维护性，类、类属性、类方法的注释必须使用 Javadoc 规范。注释格式应为 /\*\* 内容 \*/，避免使用 // xxx 的单行注释方式。Javadoc 注释应详细、准确，涵盖类的功能、属性含义、方法参数、返回值以及可能抛出的异常等信息。
10. 抽象方法注释：所有的抽象方法（包括接口中的方法）除了按照 Javadoc 规范进行注释外，还需特别强调其实现的功能。除了描述方法的返回值、参数和可能抛出的异常外，还应明确指出该方法的主要作用和执行的操作，以便调用者正确理解和使用。
11. **创建者信息**

为了便于代码追踪和团队协作，所有的类都必须在适当的位置添加创建者信息。这有助于明确代码的维护责任，并在必要时联系相关人员进行咨询或修改。

1. **内部注释**
2. 单行注释：在方法内部，若需要对某行代码进行简短说明，应使用单行注释。注释应位于被注释语句的上方，使用 // 开头，确保注释与代码对齐，便于阅读。
3. 多行注释：对于需要跨越多行的注释内容，应使用 /\* \*/ 格式的多行注释。同样，注释应与代码保持对齐，以提高代码的可读性。
4. **安全规约**
5. **用户敏感数据保护**

为确保用户隐私和数据安全，所有敏感数据在展示前必须进行脱敏处理。任何直接展示用户敏感数据的行为都是严格禁止的，脱敏方法应根据数据类型和敏感程度进行合理选择，以平衡数据的安全性和可用性。

1. **SQL注入防护**

为防范SQL注入攻击，用户输入的SQL参数必须严格使用参数绑定或METADATA字段值限定。禁止通过字符串拼接的方式构造SQL语句来访问数据库，以减少潜在的安全风险。

1. **参数有效性验证**

对于用户请求传入的任何参数，必须进行严格的有效性验证。这包括但不限于检查参数的数据类型、长度、格式以及是否满足业务逻辑要求。任何未经验证或验证失败的参数都不应被系统接受或处理，以防止潜在的安全漏洞和错误。

1. **资源控制**

在使用平台资源，如发送短信、邮件、电话、下单、支付等操作时，必须实现正确的防重放限制措施。这包括但不限于数量限制、疲劳度控制以及验证码校验等机制。通过这些措施，可以有效防止恶意用户滥用平台资源，保护平台免受经济损失和声誉损害。

1. **数据库规约**
2. **命名规范**
3. 数据表与字段命名：表名和字段名必须使用小写字母或数字进行命名，以确保跨平台的一致性和兼容性。禁止出现以数字开头的命名，以避免潜在的解析错误。同时，禁止在两个下划线中间仅出现数字，以保持命名规则的清晰和简洁。
4. 数据表命名：表名应反映其所包含数据的主要内容或实体，且不应使用复数名词。使用单数形式有助于在代码中更准确地表达单个记录的概念，避免歧义。
5. 是与否概念的字段命名：为了统一表达逻辑上的肯定与否定概念，涉及此类概念的字段必须使用 is\_xxx 的命名方式。数据类型应选用 int 类型，其中 1 表示肯定（是），0 表示否定（否）。这种命名方式有助于增强代码的可读性和维护性。
6. **禁用保留字**

在数据库设计中，应禁用所有数据库系统的保留字，如 MySQL 中的 desc、range、match、delayed 等。使用保留字可能导致语法错误或意外的行为，因此应参考各数据库官方文档中的保留字列表，确保表名、字段名等不与保留字冲突。

1. **必备字段**

每张数据表都应包含以下三个必备字段：id（唯一标识符）、create\_time（记录创建时间）、update\_time（记录最后更新时间）。这些字段有助于跟踪数据的生命周期和变化历史，是数据库设计中的基本要素。此外，建议添加 is\_deleted 字段以标记记录的逻辑删除状态，而不是直接删除记录，以便于数据恢复和审计。

1. 超过三个表禁止join。需要join的字段，数据类型保持绝对一致；多表关联查询时， 保证被关联的字段需要有索引。
2. 不得使用外键与级联，一切外键概念必须在应用层解决。
3. 数据订正（特别是删除或修改记录操作）时，要先select，避免出现误删除，确认无 误才能执行更新语句。
4. 在表查询中，一律不要使用 \* 作为查询的字段列表，需要哪些字段必须明确写明
5. 更新数据表记录时，必须同时更新记录对应的update\_time字段值为当前时间
6. **工程结构**

本项目采用Spring Boot框架进行开发，整体架构清晰，层次分明，便于开发与维护。项目共分为View层、Controller层、Service层、Mapper层、Model层，各层之间职责明确，协同工作，确保系统的稳定与高效运行。

1. View层

View层主要负责与用户进行交互，展示数据并接收用户的输入。在本项目中，View层采用前后端分离的设计思路，前端页面通过HTTP请求与后端进行通信，获取数据并展示给用户。View层使用HTML、CSS、JavaScript等技术进行页面开发，确保界面的美观与易用性。同时，View层还负责处理用户的输入，包括表单提交、按钮点击等操作，并将用户的请求发送给Controller层进行处理。

1. Controller层

Controller层作为项目的入口点，负责接收来自View层的请求，并调用相应的Service层方法进行处理。Controller层的主要职责是解析请求参数、验证参数的有效性、调用Service层方法并返回处理结果。在返回结果时，Controller层还会根据需要进行数据格式的转换，如将Java对象转换为JSON格式，以便前端能够正确解析。Controller层通过注解和配置文件实现路由映射，确保请求能够正确地被分发到对应的处理方法上。

1. Service层

Service层是项目的业务逻辑处理层，负责实现具体的业务功能。Service层接收来自Controller层的请求，并调用Mapper层的方法获取数据或执行相应的数据库操作。在Service层中，我们可以对业务逻辑进行封装和复用，提高代码的可维护性和可扩展性。同时，Service层还可以进行数据的校验、转换和封装等操作，确保数据的准确性和一致性。Service层通常不直接操作数据库，而是通过Mapper层与数据库进行交互。

1. Mapper层

Mapper层是项目的数据访问层，负责与数据库进行交互。Mapper层使用MyBatis等持久层框架，通过XML配置文件或注解的方式定义SQL语句，实现对数据库的增删改查操作。Mapper层接收Service层的请求，并根据请求调用相应的SQL语句执行数据库操作。Mapper层返回的结果通常是Java对象或集合，方便Service层进行数据处理。Mapper层的存在使得业务逻辑与数据访问解耦，提高了代码的可读性和可维护性。

1. Model层

Model层是项目的数据模型层，定义了项目中使用的所有实体类。这些实体类通常与数据库中的表对应，包含了表的字段信息以及相关的getter和setter方法。Model层的主要作用是提供数据的封装和传输功能，使得数据在不同层之间传递时能够保持一致性。同时，Model层还可以定义数据校验规则，确保数据的合法性和有效性。