在线车票服务系统编程规范

2025年7月10日

本编程规范旨在为Mini-12306项目的后端开发提供一套统一的代码编写准则，以提高代码的可读性、可维护性、可测试性，并促进组员之间的协作效率。遵循统一的编程规范对于构建高质量、可持续发展的软件系统至关重要。

1. 核心原则：

一致性：在整个项目中保持代码风格和约定的高度一致性。如果因特殊原因无法遵循某条规则，应添加简要注释说明。

可读性：代码应清晰易懂，如同良好书写的散文，使其他一起开发的组员能够快速理解其意图和逻辑。

可维护性：编写易于修改和扩展的代码，减少引入新错误的风险。

1. 主要参考规范

本项目的Java后端代码将主要遵循 Google Java Style Guide。所有未在此文档中明确提及的特定风格问题，均以 Google Java Style Guide 为准。

建议团队成员熟悉并使用IDE（如IntelliJ IDEA或VSCode）中针对Google Java Style Guide的格式化插件和检查工具（如Checkstyle），以自动化规范的执行。

1. 核心编程约定

以下是基于Google Java Style Guide并结合项目特点强调的关键约定：

* 1. 命名约定 (Naming Conventions)

包 (Packages)：全部小写，使用.分隔。例如：com.example.techprototype.controller。

类 (Classes) / 接口 (Interfaces) / 枚举 (Enums) / 注解 (Annotations)：大驼峰命名法 (PascalCase)。例如：ProfileController, MyTicketResponse, OrderStatus。

方法 (Methods)：小驼峰命名法 (camelCase)。例如：getUserProfile, updateUserProfile, getTicketDetail。

变量 (Variables)：小驼峰命名法 (camelCase)。局部变量、实例变量均适用。例如：userId, orderNumber, availableStock。

常量 (Constants)：全部大写，单词之间用下划线 \_ 连接。例如：MAX\_PASSENGERS, DEFAULT\_PAGE\_SIZE。

类型参数 (Type Parameters)：单个大写字母，通常是T, E, K, V, S等。

* 1. 代码格式化 (Code Formatting)

缩进 (Indentation)：使用4个空格进行缩进，最好别使用Tab。

括号 (Braces)：采用K&R风格，即左大括号 { 不换行，右大括号 } 换行。

public class MyClass {

        public void myMethod() {

            if (condition) {

                // ...

            } else {

                // ...

            }

        }

    }

行长度 (Line Length)：每行代码的长度建议不超过120个字符。超过时应考虑换行。

空格 (Whitespace)：

二元运算符前后应有空格（+, -, \*, /, =, ==, <, > 等）。

关键字（if, for, while等）后跟括号时，应有空格。例如：if (condition)。

逗号 , 和分号 ; 后应有空格。

* 1. 注释 (Commenting)

正如一些软件标准文档中强调的，文档是软件开发不可或缺的一部分。

文件注释 (File Comments)：每个Java源文件顶部最好包含一个文件注释块，简要描述文件的目的和功能。

类注释 (Class Comments)：所有公共类都应包含Javadoc注释，说明类的目的、职责。

方法注释 (Method Comments)：所有公共方法（包括Controller、Service、DAO层的方法）都应包含Javadoc注释，说明方法的功能、参数 (@param)、返回值 (@return)、可能抛出的异常 (@throws)。对于复杂的业务逻辑，应详细说明其处理流程。

行内注释 (Inline Comments)：用于解释非显而易见的逻辑、算法选择或绕过某些限制的代码。应简洁明了。避免注释显而易见的或代码本身就能说明的问题。

TODO注释 (TODO Comments)：使用// TODO: [描述] 标记待完成、待优化或待修复的代码点，以便后续跟踪。

* 1. 异常处理 (Exception Handling)

具体化异常：捕获特定异常，而不是笼统地捕获 Exception。

统一响应：后端API应提供统一的错误响应格式（如本次迭代中使用的BasicResponse或ProfileResponse中的failure方法），包含状态、消息和时间戳，便于前端处理。

业务异常：对于可预期的业务逻辑错误，应定义自定义业务异常，并在Service层抛出，在Controller层统一捕获并转换为友好的错误信息。

日志记录：在捕获异常时，最好记录一下详细的错误日志，包含堆栈信息，以便排查问题。

* 1. 日志记录 (Logging)

使用SLF4J + Logback：项目中已集成，应统一使用org.slf4j.Logger进行日志输出。

日志级别：根据信息重要性选择合适的日志级别：

ERROR：系统运行时错误，如数据库连接失败，未捕获的异常。

WARN：潜在问题或不推荐的做法，但不影响系统正常运行。

INFO：系统关键操作或状态变化，如服务启动、请求处理开始/结束。

DEBUG：开发或调试时使用的详细信息。

TRACE：最详细的，通常用于跟踪方法调用流程。

日志内容：日志应包含足够的信息，如请求参数、关键业务ID（用户ID、订单号）、异常信息等。避免记录敏感信息（如密码、身份证号）。

* 1. 代码结构与模块化 (Code Structure and Modularity)

分层架构：严格遵循Controller-Service-DAO（Repository）三层架构：

Controller：负责接收请求、参数校验、调用Service层，并构建HTTP响应。不应包含复杂业务逻辑。

Service：封装业务逻辑、事务管理、协调多个DAO操作。

DAO/Repository：负责与数据库交互，进行数据持久化操作。

职责单一原则：每个类和方法都应只负责一个明确的功能。

DTO (Data Transfer Object)：用于数据传输，尤其是在Controller和Service层之间，以及Service层和外部系统（如前端）之间。避免直接暴露Entity对象。

* 1. 依赖管理 (Dependency Management)

Maven：所有项目依赖均通过pom.xml进行管理。

版本控制：明确指定依赖的版本，避免使用LATEST或不稳定的版本。

按需引入：只引入项目实际需要的依赖，避免引入不必要的库增加项目体积和潜在冲突。

* 1. 版本控制 (Version Control)

Git：项目使用Git进行版本控制。

分支策略：建议采用git flow主流分支管理策略。

提交信息：提交信息应清晰、简洁、有意义，说明本次提交的目的和内容。例如：feat: 实现用户注册功能，fix: 修复个人信息更新bug。

正如软件标准文档中提到的，版本控制工具（如CVS，我们组本项目对应Git）对于团队协作和版本回溯至关重要。

1. 特定技术栈约定

Spring Boot：

使用@RestController和@RequestMapping定义RESTful API。

合理使用Spring的DI（依赖注入）和IoC容器。

事务管理使用@Transactional注解，确保原子性操作。

Spring Data JPA：

利用JPA Repository接口简化数据访问层代码。

对于复杂查询，优先使用JPQL的@Query注解，或考虑Specification/Criteria API。

注意N+1查询问题，合理使用@EntityGraph或JOIN FETCH。

Redis：

统一通过RedisService或类似封装类进行操作，避免直接在业务代码中暴露RedisTemplate。

库存相关的Key命名遵循约定（如：stock:{trainId}:{departureStopId}:{arrivalStopId}:{travelDate}:{carriageTypeId}）。

分布式锁使用Redisson。

RabbitMQ：

消息生产者和消费者应通过统一的服务或组件进行封装，避免业务代码直接与MQ API耦合。

定义明确的消息格式。

1. 性能与安全考量

性能优化：

数据库查询应考虑索引优化，避免全表扫描。

对于频繁访问且变化不大的数据，考虑使用Redis缓存。

高并发核心路径（如购票）应特别注意原子性操作和异步处理。

座位区间占用：明确采用二进制位操作的思想，确保高效判断和存储区间占用信息，避免数据冗余和性能瓶颈。

安全：

密码加密：使用BCryptPasswordEncoder等安全算法对密码进行哈希存储。

权限验证：API接口应有严格的权限验证（如：车票详情页的乘客ID一致或订单用户ID一致的逻辑）。

数据脱敏：敏感信息（如身份证号）在返回给前端或日志记录时最好进行脱敏处理。

输入校验：对所有用户输入进行严格的校验，防止SQL注入、XSS等攻击。