开发规范

2026年9月20日 17:04

第一部分 开发代码规范

jar包版本：2.2.1

一、项目结构与分层规范

1.1 标准分层架构

严格遵循 **Controller → Service → Mapper** 三层架构，**禁止跨层调用**。（支持模块化拆分；若为单模块项目，可省略模块名层级。）

```
src/main/java/com/jiangh/module/
├── annotation/              # 注解
├── config/                  # 配置
├── common/                  # 工具
├── controller/              # Controller，仅处理 HTTP 关注点
├── converter/               # 转换器
├── enums/                   # 枚举
├── service/                 # Service，业务逻辑 + 事务边界
│   └── impl/                 # 服务实现
├── mapper/                  # MyBatis-Plus Mapper 接口
├── handler/                 # 处理器
├── interceptor/             # 拦截器
├── listener/                # 监听器
├── entity/                  # 实体
│   ├── request/              # 请求实体
│   ├── response/             # 响应实体
│   └── dto/                  # 内部传输 DTO
└── event/                   # ApplicationEvent 事件与监听器
```

1.2 包组织原则

- **推荐按功能切片（feature slice）组织包**，而非按技术层平铺。每个功能模块内自包含 Controller、Service、Mapper、DTO。
- 每个 `@Configuration` 类**只负责一个关注点。**
- **不允许**自定义配置类覆盖系统包的配置类。
- **禁止**将所有类堆在一个包下（例如财务系统常见的"一包到底"）。应按功能区分，例如：`src/main/java/com/jiangh/module/baseData`。
- **建议新项目使用 Record（记录类）替代 DTO。**

1.3 分层职责

层	职责	禁止事项
Controller	HTTP 参数校验、状态码、响应映射	禁止包含业务逻辑

Service	业务逻辑、事务边界、领域规则	禁止直接操作 `HttpServletRequest`
Mapper	数据访问	禁止包含业务逻辑
entity	请求 / 响应数据传输	—

二、命名规范

元素	规范	示例
类名	PascalCase	`UserController`、`ISystemConfigService`
方法名 / 变量名	camelCase	`findUserById`、`isOrderValid`
常量	UPPER_SNAKE_CASE	`MAX_RETRY_ATTEMPTS`、`DEFAULT_PAGE_SIZE`
包名	全小写，点分隔	`com.jiangh.system`
Entity	PascalCase	`ChangePasswordRequest`、`MenuMetaResponse`
数据库表	snake_case，功能模块开头	`system_user`

三、依赖注入规范

3.1 构造函数注入（强烈建议）

必须使用**构造函数注入**，**禁止字段注入**（`@Autowired` 标注在字段上）。 构造函数注入保证不可变性、可测试性，并显式暴露依赖关系。

```
// ☑正确：构造函数注入 + final
@Service
@RequiredArgsConstructor
public class UserService {
    private final UserRepository userRepository;
    private final EventPublisher eventPublisher;
}

// ✕禁止：字段注入
@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;  // 非 final，难以测试
}
```

四、REST API 设计规范

4.1 HTTP 方法与状态码

- 正确使用 HTTP 方法：`GET`（读取）、`POST`（创建）、`PUT`（全量更新）、`PATCH`（部分更新）、`DELETE`（删除）。

4.2 请求与响应

- 请求 / 响应 DTO **禁止直接返回 JSON 裸串**，必须使用固定的实体类。
- 请求体使用 Jakarta Bean Validation：`@Valid`、`@NotBlank`、`@Size`、`@Email` 等。
- **系统响应必须使用系统包内的 `Result<T>` 类**，删除项目内自定义的响应封装类。
- 对重要操作接口（数据修改、删除、新增等）必须添加日志注解：

```
@LogJiangh(module = WaterModuleConstant.waArea, method = "新增、修改")
```

属性	说明

method	操作方法
module	模块名
params	是否记录请求参数，默认 true
result	是否记录响应结果，默认 false

- 对极其重要的接口需添加权限校验，进行二次校验：

  `@PreAuthorize("@ss.hasPerm('EnableHandControl')")`

  说明：`EnableHandControl` 为菜单管理内设置的按钮权限。

- 对第三方接口需要使用加 / 解密注解对接口进行加 / 解密，**不允许明文传输**（3des：默认、sm4、rsa）：

encType 值	加密算法	签名算法
JiangHConstants.DES	DESede/ECB/PKCS5Padding	—
JiangHConstants.SM4	SM4/CBC/PKCS5Padding	—
JiangHConstants.RSA	RSA/ECB/PKCS1Padding	MD5withRSA

```
@Encrypt(encType = JiangHConstants.DES)
@Decrypt(encType = JiangHConstants.DES, required = false, timeout = 300)
```

属性	说明
encType	加 / 解密类型
required	是否必须加密，默认 true
timeout	请求数据时间戳校验时间差

- 对开放接口**必须携带固定 token**，且token需要在 METER_FACTORY_TOKE 内配置

4.3 API 文档

使用 SpringDoc OpenAPI 注解（`@Operation`、`@Tag`、`@ApiResponse`）生成 API 文档。

五、异常处理规范

5.1 全局异常处理

- **统一由系统包处理。**
- **禁止**自定义 `@ControllerAdvice` + `@ExceptionHandler`。

5.2 异常设计原则

- 对有预期的错误，抛出 `BaseException`。
- 其他异常建议使用 `RuntimeException`。
- 注意：若抛出 `BaseException` 之外的异常，会导致响应状态码非 `200`。
- 对第三方提供的接口需返回明确的状态码，**禁止抛出异常**。
- **不允许**自定义异常类。

六、事务管理规范

- 在 Service 类级别标注 `@Transactional(readOnly = true)`，写操作方法单独标注 `@Transactional`。
- 事务方法**必须为** `public`，同类内部调用不会触发事务代理。
- 受检异常默认不触发回滚，需显式声明：`@Transactional(rollbackFor = IOException.class)`。

- 以**业务动作**为单位划分事务边界，而非以技术框架为单位。

七、安全规范

7.1 密码与加密

```
private final PasswordEncoder passwordEncoder;
```

- 修改密码、加密统一使用 `passwordEncoder.encode(password)`。
- 密码强度校验使用 `PasswordUtil.checkPasswordStrength(password)`，返回结果不能小于 `6`。
- 密码使用 `BCryptPasswordEncoder`。

7.2 通用安全规则

- 禁止在配置文件中硬编码密钥，使用环境变量引用 `${JIANGH_WATER_DB_PASSWORD}`。**强烈建议。**
- 生产环境设置 `server.error.include-message=never`，避免泄露内部错误信息。
- **生产环境必须关闭文档接口。**
- Nginx 隐藏版本号，在 `http` 块中添加：`server_tokens off;`

八、日志规范

- 统一使用 **SLF4J** 作为日志门面，Logback 作为实现（Spring Boot 3 默认组合）。
- 使用 Lombok `@Slf4j` 注入 Logger，**仅允许使用 Lombok 的 `@Slf4j`**。
- **必须使用占位符**，禁止字符串拼接：

```
// ☑ 正确
log.info("User found: id={}, name={}", userId, userName);

// ✕ 禁止
log.info("User found: id=" + userId + ", name=" + userName);
```

日志级别使用规范：

级别	使用场景
ERROR	系统错误，需告警
WARN	可恢复异常
INFO	关键业务流程节点
DEBUG	调试信息，生产环境关闭

九、虚拟线程使用规范

9.1 启用方式

Spring Boot 3.2+ 通过一行配置启用虚拟线程，Tomcat、`@Async`、定时任务等将自动使用：

```
spring.threads.virtual.enabled=true
```

9.2 使用原则

- 虚拟线程适用于 **I/O 密集型任务**（数据库查询、HTTP 调用、Redis 操作、文件 I/O），**不适用于 CPU 密集型任务**。
- **禁止池化虚拟线程**：创建成本极低，每次任务新建即可，池化会限制其核心优势。
- 需要限制并发时，使用 `Semaphore`、连接池或限流器，而非固定大小的虚拟线程池。

- 高频路径上**避免持有 `synchronized` 锁执行慢 I/O**，可能导致载体线程被固定（Pinning），优先使用 `ReentrantLock`。
- 避免在虚拟线程中使用 `ThreadLocal` 存储大对象，优先使用 `ScopedValue`（JDK 21 预览特性，为虚拟线程设计）。
- 并行独立操作使用 `CompletableFuture`，结构化并发可使用 `StructuredTaskScope`。

十、工具类使用规范（强制规范）

10.1 HTTP 请求

必须使用系统包内的 `HttpUtil` 或 `OkHttpUtil`，删除项目内自建的工具类：

工具类	说明
`HttpUtil`	基于 HttpClient5 的常规请求
`OkHttpUtil`	基于 OkHttp，带两次重试机制、不验证证书

10.2 IP 工具

`IPUtil`：获取内网 IP、真实请求 IP，以及解析地址、区域。
对区域限制的系统需要定期更新 IP 库：
https://github.com/lionsoul2014/ip2region/tree/master/data

10.3 Redis 工具

使用系统包内的 `RedisTool`，**删除项目内自建的工具类**。

10.4 文件工具

使用 `FilesUtilJH`，或使用 `FileUploadService` 的实现类 `localFileUploadService`（本地服务）、`ctyunFileUploadService`（OSS）。

10.5 日志

统一使用 `@Slf4j` 下的 `log` 输出日志。

10.6 字典表

- **统一使用系统包内的字典表**（数据量大、带功能性数据不建议使用字典表）。
- 需要翻译的字段使用 `@Dict` 注解。
- 如果是未登录情况下，需要设置：`SecurityUtils.setSystemId(systemId, "定时任务")`。

属性	说明	默认值
`dictTable`	对应的表	字典表
`dicCode`	对应的字段	字典表内字段
`dictColumn`	最终翻译的字段	—
`dicText`	需要翻译的字段	原字段名 + `DictText`

示例 1：将 `departmentId` 翻译为 `co_department` 表内 `id` 对应值的 `department_name`，最终字段为 `department_name`：

```
@Dict(dictTable = "co_department", dicCode = "id",
      dictColumn = "department_name", dicText = "department_name")
private String departmentId;

@TableField(exist = false)
@Schema(description = "部门翻译字段")
private String departmentName;
```

示例 2：将 `isDepreciation` 翻译为字典表对应 `code = is_depreciation` 的值，赋值给 `isDepreciationDictText`：

```
@Schema(description = "是否折旧 0：不折旧 1：折旧")
@Dict(dicCode = "is_depreciation")
private String isDepreciation;

@Schema(description = "是否折旧翻译字段")
private String isDepreciationDictText;
```

其余逻辑请咨询翔总，本规范基于翔总字典表并修复 BUG 后的版本。

10.7 模板类消息

建议使用**微信工具类**进行配置并生成对应的格式：

- `wxService.buildKeywordJson`：生成 Keyword 格式

```
"userid":{ "value":"39724" },
"userName":{ "value":"逆龄**********" },
"userAddress":{ "value":"发达广场D区横店影城地下负一层游泳馆" }
```

- `wxService.buildJson`：生成 json 格式（`userId:"2001",account:"2025/01",arreas:"200"`）
- `wxService.buildArray`：生成数组格式，只有值没有 key，如 `[2001, "2025/01", "200"]`

10.8 防止重复提交注解

```
@RepeatSubmit(expire = 600, unlock = true)
```

属性	说明	默认值
`expire`	锁过期时间	5 秒
`unlock`	执行完立即解锁	`false`（等待锁自动过期）
`validateParams`	是否校验参数	`false`（操作员 + 请求路径 + 操作方法）

10.9 校验字段是否为指定值

```
@ValidField(allowedValues = {"STRING"})
```

当前字段仅允许传：`STRING`。

10.10 加密注解

将对应字段的数据加密存储到数据库，查询后解密。注意此字段**无法模糊查询**，需要查询时使用 sharding 加密。

```
@JasyptField
```

10.11 配置信息

大部分移到系统配置，各项目也可以使用此配置信息：

```
private final ISystemConfigService systemConfigService;
```

- 获取全局配置：`configService.getGlobalSystemConfig(key);`
- 指定区域系统配置：`String systemConfig = systemConfigService.getSystemConfig(key, systemId);`

属性	说明	默认值
`key`	配置的 key	—
`systemId`	区域信息	—

10.12 其他工具类

其余工具类建议使用 **Hutool**。

附：开发核心约定速查

维度	强制约定
分层	Controller → Service → Mapper，禁止跨层
注入	构造函数注入 + `final`
响应	统一使用系统 `Result<T>`
异常	预期错误抛 `BaseException`，禁止自定义异常类
日志	`@Slf4j` + 占位符
工具类	优先用系统包，禁止重复造轮子
密码	`BCryptPasswordEncoder` + `PasswordUtil` 强度校验
文档	生产环境关闭 Swagger
虚拟线程	I/O 密集型适用，禁止池化

第二部分　代码提交规范

一、分支策略

采用**简化 Git Flow**，核心分支与辅助分支职责明确：

分支类型	命名格式	来源	合并目标	说明
主分支	`main` / `master`	—	—	**保护分支**，仅允许通过 PR/MR 合并，不允许直接提交
开发分支	`develop`	`main`	`main`（通过release）	日常开发集成分支
功能分支	`feature/{模块名}-{简述}`	`develop`	`develop`	例：`feature/user-login`
修复分支	`fix/{模块名}-{简述}`	`develop`	`develop`	例：`fix/payment-timeout`
紧急修复	`hotfix/{模块名}-{简述}`	`main`	`main` + `develop`	例：`hotfix/login-500`
发布分支	`release/v{主版本}.{次版本}`	`develop`	`main` + `develop`	例：`release/v1.2`

核心规则：

- `main` 和 `develop` 必须设置分支保护，所有 CI 检查必须通过。
- feature/fix 分支生命周期尽量短，**建议不超过 3 天**。
- 分支合并后及时删除远程分支。

二、提交信息格式（Conventional Commits）

所有提交信息**必须**遵循以下格式：

```
<type>(<scope>): <subject>
```

type	含义	示例
`feat`	新功能	`feat(user): 新增用户批量导入接口`
`fix`	修复 bug	`fix(order): 修复订单金额计算精度丢失`
`docs`	文档更新	`docs(api): 更新用户接口文档`
`style`	代码格式	`style(frontend): 统一缩进为 2 空格`
`refactor`	重构	`refactor(service): 抽取公共分页逻辑`

`test`	测试相关	`test(user): 补充用户注册单元测试`
`chore`	构建/工具链/依赖	`chore(deps): 升级 Spring Boot 到 3.2.5`
`perf`	性能优化	`perf(query): 优化订单列表查询索引`

scope（可选）：影响范围，如 `user`、`order`、`frontend`、`backend`、`db`。

subject（必填）：不超过 50 字符，动词开头，中文即可。

```
git commit -m "feat(user): 新增微信登录支持"
git commit -m "fix(order): 修复购物车数量计算错误"
git commit -m "chore(db): 添加订单表联合索引迁移脚本"
```

强制规则：冒号后必须有空格；同一提交只做一件事；提交前必须通过本地 lint / 格式化。

三、提交前检查清单

检查项	前端（Vue）	后端（Spring Boot）
代码格式	ESLint + Prettier 通过	Spotless / Checkstyle 通过
静态检查	`vue-tsc` 无类型错误	SpotBugs 无高危问题
单元测试	相关组件测试通过	相关 Service 测试通过
敏感信息	无 `.env` 密钥提交	无数据库密码硬编码
数据库脚本	—	Flyway 迁移脚本已加入 `db/migration/`

四、Pull Request / Merge Request 规范

- **PR 大小**：单个 PR 变更建议**不超过 400 行**。
- **PR 标题**：必须遵循 Conventional Commits 格式。
- **PR 描述**：说明"为什么做"和"怎么做"，关联 Issue（如 `Closes #123`），UI 变更附截图。
- **审查者**：至少 1-2 人，优先邀请相关模块负责人。
- **审查重点**：逻辑正确性、命名规范、安全漏洞（SQL 注入 / XSS）、数据库迁移脚本是否幂等。
- **合并方式**：feature → develop 用 **Squash Merge**；develop → main 用 **Merge Commit**。

五、Git Hooks 强制校验

Hook	工具	作用
`pre-commit`	Husky + lint-staged（前端）	提交前自动格式化
`commit-msg`	commitlint + Maven/Gradle 插件	校验提交信息是否符合规范
`pre-push`	自定义脚本	阻止直接 push 到 `main` / `develop`

CI 兜底：在流水线中增加 `commitlint` 检查步骤，即使本地 Hook 被绕过，CI 也会拒绝合并。

第三部分　部署规范

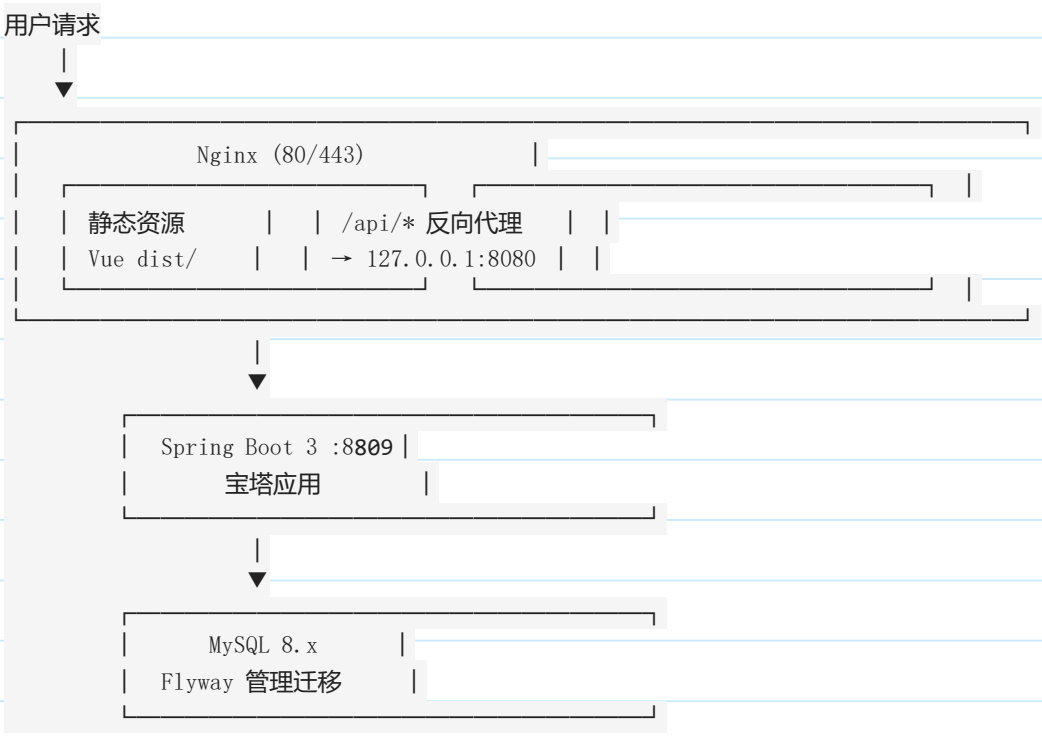
一、环境规划

环境	用途	前端	后端	数据库
开发（dev）	本地开发	`npm run dev` (Vite)	IDE 启动	本地 MySQL

生产（prod）	正式环境	Nginx + HTTPS	宝塔面板网站配置	主从 + 定期备份

核心原则：所有环境**仅通过 Nginx 统一对外暴露 端口**，后端服务监听 `127.0.0.1:8080`，禁止直接暴露后端端口。

二、整体部署架构



三、前端（Vue）部署规范

3.1 构建

```
# 在 frontend/ 目录下
npm ci                    # 严格按 package-lock.json 安装
npm run build             # Vite 构建，产出 dist/
```

生产环境变量在 `.env.production` 中配置，**禁止将 API 地址硬编码在代码里**：

```
# .env.production
VITE_API_BASE_URL=/api
```

3.2 Nginx 配置（前端核心）

其余为宝塔面板默认配置

```
# 前端静态资源
    root /var/www/vue-app/dist;
    index index.html;

# http块内配置
map $http_upgrade $connection_upgrade {
default upgrade;
'' close;
}

# API 反向代理到后端
    location /reportServiceApi {

      proxy_pass  http://reportServer/reportServiceApi;
      proxy_set_header   Host $host:$server_port;
```

```
            proxy_set_header   X-Real-IP $remote_addr;
            proxy_set_header   X-Forwarded-For  $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header Upgrade $http_upgrade;
     proxy_set_header Connection $connection_upgrade;
     proxy_read_timeout 500000s;
    }
}
```

四、后端（Spring Boot 3）部署规范

4.1 生产环境配置（application-prod.yml）

```
server:
  port: 8080
  address: 127.0.0.1                # 仅监听本地，由 Nginx 转发
  error:
    include-message: never          # 不泄露内部错误信息

spring:
  threads:
    virtual:
      enabled: true                 # 启用虚拟线程
  datasource:
    url: jdbc:mysql://${JIANGHONG_DB_HOST}:3306/${JIANGHONG_DB_NAME}?useSSL=true&serverTimezone=Asia/Shanghai
    username: $JIANGHONG_{DB_USERNAME}
    password: ${JIANGHONG_DB_PASSWORD}        # 从环境变量注入，禁止硬编码
    hikari:
      maximum-pool-size: 20
      minimum-idle: 5
  flyway:
    enabled: true
    locations: classpath:db/migration
    baseline-on-migrate: true

# 生产环境关闭文档
springdoc:
  api-docs:
    enabled: false
  swagger-ui:
    enabled: false
```

环境变量 以JIANGHONG_模块名 开始，通用的可以不加模块名

如：自来水的数据库地址：JIANGHONG_WATER_DB_PASSWORD

五、MySQL 数据库部署规范

5.1 添加表字段

/resources/db/修改表字段.xls

所有数据库添加字段需要添加到这个文件内，文件格式

表名	字段名	语句	附加语句	类型
	字段名或索引名	要执行的SQL	创建字段后需要同时执行的SQL，多个以逗号隔开，如添加索引、设置值・・・	字段（COLUMN）索引（INDEX）

5.2新建表

/resources/db/schema.sql
application.properties 配置:
spring.datasource.dynamic.datasource.master.init.schema=db/schema.sql
禁止事项:

- **禁止**在代码中硬编码数据库密码，统一使用环境变量 `${JIANGH_DB_PASSWORD}`。

5.3 备份策略

策略	频率	保留	工具
全量备份	每日 02:00	30 天	`mysqldump` / Percona XtraBackup
增量备份	每 6 小时	7 天	binlog
备份验证	每周一次	—	恢复至测试库并校验数据

六、CI/CD 流水线

以 GitLab CI 为例:

```
stages:
  - lint          # 代码检查 + 提交信息校验
  - test          # 单元测试 + 集成测试
  - build         # 前端构建 + 后端打包
  - migrate       # Flyway 数据库迁移
  - deploy        # 部署到目标环境

commitlint:
  stage: lint
  script:
    - npx commitlint --from $CI_MERGE_REQUEST_DIFF_BASE_SHA --to HEAD

build:frontend:
  stage: build
  script:
    - cd frontend && npm ci && npm run build
  artifacts:
    paths: [frontend/dist/]

build:backend:
  stage: build
  script:
    - cd backend && ./gradlew bootJar
  artifacts:
    paths: [backend/build/libs/*.jar]

migrate:
  stage: migrate
  script:
    - cd backend && ./gradlew flywayMigrate
  environment:
    name: staging
  rules:
    - if: $CI_COMMIT_BRANCH == "develop"
```

部署门禁:

- `main` 分支的部署必须经过**人工审批**。
- 数据库迁移失败时，**自动阻止部署**。

七、生产环境安全配置

项目	配置
HTTPS	建议 HTTPS，Nginx 配置 301 跳转
后端暴露	后端仅监听 `127.0.0.1:8080`，不对外暴露
文档	生产环境关闭 Swagger / SpringDoc
Nginx 版本	隐藏版本号：`server_tokens off;`
数据库密码	环境变量注入，禁止硬编码
错误信息	`server.error.include-message=never`
日志	生产环境日志级别设为 `INFO`，禁止输出 DEBUG
备份	数据库每日全量 + binlog 增量备份

附：部署核心约定速查

维度	强制约定
入口	仅 Nginx 暴露 端口，后端监听 127.0.0.1
前端	`try_files` 解决 history 404，`client_max_body_size` ≥ 20m
数据库	Flyway 版本化迁移，禁止 `ddl-auto=update`
安全	生产关文档、隐藏 Nginx 版本、环境变量注入密码
备份	每日全量 + binlog 增量