# **教程平台软件配置与运维文档**

**-zxy**

## **文档概述**

本文档详细描述了教程平台项目的配置管理、版本控制、持续集成、部署和运维计划，为项目的稳定运行和团队协作提供指导。

## **1. 配置管理**

### **1.1 环境配置策略**

#### **1.1.1 多环境配置**

plaintext

├── application.properties # 基础配置

├── application-dev.properties # 开发环境配置

├── application-test.properties # 测试环境配置

└── application-prod.properties # 生产环境配置

#### **1.1.2 配置项分类**

****数据库配置****

properties

# 开发环境spring.datasource.url=jdbc:mysql://localhost:3306/tutorial\_platform\_devspring.datasource.username=dev\_userspring.datasource.password=${DB\_PASSWORD:dev123}

# 生产环境spring.datasource.url=jdbc:mysql://prod-db:3306/tutorial\_platformspring.datasource.username=${DB\_USER:prod\_user}spring.datasource.password=${DB\_PASSWORD}

****应用配置****

properties

# 服务端口配置server.port=${PORT:8088}

# 文件上传配置file.upload-dir=${UPLOAD\_DIR:upload\_data}spring.servlet.multipart.max-file-size=${MAX\_FILE\_SIZE:10MB}

# JWT配置jwt.secret=${JWT\_SECRET:defaultSecret}jwt.expiration=${JWT\_EXPIRATION:86400}

### **1.2 配置管理工具**

#### **1.2.1 Spring Boot Configuration Processor**

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional></dependency>

#### **1.2.2 配置验证**

java

@ConfigurationProperties(prefix = "app")@Validatedpublic class AppProperties {

@NotBlank

private String name;

@Min(1)

@Max(65535)

private int port;}

### **1.3 敏感信息管理**

#### **1.3.1 环境变量方式**

bash

export DB\_PASSWORD=your\_secure\_passwordexport JWT\_SECRET=your\_jwt\_secret\_keyexport REDIS\_PASSWORD=your\_redis\_password

#### **1.3.2 配置文件加密**

* 使用 Jasypt 进行配置文件敏感信息加密
* 生产环境密钥通过环境变量注入

## **2. 版本控制**

### **2.1 Git 分支策略**

#### **2.1.1 GitFlow 工作流**

plaintext

master/main # 生产分支，只包含稳定版本

├── develop # 开发分支，功能集成分支

├── feature/\* # 功能分支，从develop分出

├── release/\* # 发布分支，从develop分出

└── hotfix/\* # 热修复分支，从master分出

#### **2.1.2 分支命名规范**

plaintext

feature/user-authentication # 用户认证功能

feature/file-upload # 文件上传功能

bugfix/login-error # 登录错误修复

hotfix/security-patch # 安全补丁

release/v1.0.0 # 版本发布

### **2.2 版本号管理**

#### **2.2.1 语义化版本控制**

plaintext

版本格式：主版本号.次版本号.修订号 (MAJOR.MINOR.PATCH)

示例：

- 1.0.0 # 初始稳定版本

- 1.1.0 # 新增功能，向后兼容

- 1.1.1 # 问题修复，向后兼容

- 2.0.0 # 重大变更，可能不向后兼容

#### **2.2.2 Maven 版本管理**

xml

<version>1.0.0-SNAPSHOT</version> <!-- 开发版本 --><version>1.0.0</version> <!-- 发布版本 -->

### **2.3 代码审查**

#### **2.3.1 Pull Request 流程**

1. 创建功能分支
2. 完成开发和测试
3. 提交 Pull Request
4. 代码审查（至少一人）
5. 通过 CI/CD 检查
6. 合并到目标分支

#### **2.3.2 代码审查检查点**

* 代码规范性
* 功能正确性
* 性能影响
* 安全考虑
* 测试覆盖率

## **3. 持续集成**

### **3.1 CI/CD 流水线设计**

#### **3.1.1 GitHub Actions 配置示例**

yaml

name: CI/CD Pipeline

on:

push:

branches: [ main, develop ]

pull\_request:

branches: [ main ]

jobs:

test:

runs-on: ubuntu-latest

services:

mysql:

image: mysql:8.0

env:

MYSQL\_ROOT\_PASSWORD: test123

MYSQL\_DATABASE: tutorial\_platform\_test

options: >-

--health-cmd="mysqladmin ping"

--health-interval=10s

--health-timeout=5s

--health-retries=3

steps:

- uses: actions/checkout@v3

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

- name: Cache Maven packages

uses: actions/cache@v3

with:

path: ~/.m2

key: ${{ runner.os }}-m2-${{ hashFiles('**\*\*/pom.xml')** }}

- name: Run tests

run: mvn clean test

- name: Generate test report

run: mvn jacoco:report

- name: Upload coverage to Codecov

uses: codecov/codecov-action@v3

build:

needs: test

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v3

- name: Set up JDK 17

uses: actions/setup-java@v3

with:

java-version: '17'

distribution: 'temurin'

- name: Build application

run: mvn clean package -DskipTests

- name: Build Docker image

run: |

docker build -t tutorial-platform:${{ github.sha }} .

docker tag tutorial-platform:${{ github.sha }} tutorial-platform:latest

- name: Push to registry

if: github.ref == 'refs/heads/main'

run: |

echo ${{ secrets.DOCKER\_PASSWORD }} | docker login -u ${{ secrets.DOCKER\_USERNAME }} --password-stdin

docker push tutorial-platform:${{ github.sha }}

docker push tutorial-platform:latest

### **3.2 自动化测试策略**

#### **3.2.1 测试层次**

plaintext

├── 单元测试 (Unit Tests) # 70%

├── 集成测试 (Integration Tests) # 20%

└── 端到端测试 (E2E Tests) # 10%

#### **3.2.2 测试配置**

xml

<!-- Maven Surefire Plugin --><plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<configuration>

<includes>

<include>\*\*/\*Test.java</include>

<include>\*\*/\*Tests.java</include>

</includes>

</configuration></plugin>

<!-- JaCoCo Coverage Plugin --><plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<executions>

<execution>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>report</id>

<phase>test</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

</executions></plugin>

### **3.3 质量门禁**

#### **3.3.1 代码质量检查**

* ****静态代码分析****: SonarQube 集成
* ****代码覆盖率****: 最低 80% 覆盖率要求
* ****安全扫描****: 依赖漏洞检查
* ****性能测试****: 关键接口性能基准测试

#### **3.3.2 质量标准**

yaml

quality\_gates:

- code\_coverage: ">= 80%"

- duplicated\_lines: "< 3%"

- maintainability\_rating: "A"

- security\_rating: "A"

- reliability\_rating: "A"

## **4. 部署策略**

### **4.1 容器化部署**

#### **4.1.1 Dockerfile**

dockerfile

FROM openjdk:17-jre-slim

WORKDIR /app

COPY target/tutorial\_platform-\*.jar app.jar

EXPOSE 8088

HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \

CMD curl -f http://localhost:8088/actuator/health || exit 1

ENTRYPOINT ["java", "-jar", "app.jar"]

#### **4.1.2 Docker Compose**

yaml

version: '3.8'

services:

app:

image: tutorial-platform:latest

ports:

- "8088:8088"

environment:

- SPRING\_PROFILES\_ACTIVE=prod

- DB\_HOST=mysql

- DB\_PASSWORD=${DB\_PASSWORD}

depends\_on:

- mysql

- redis

restart: unless-stopped

mysql:

image: mysql:8.0

environment:

- MYSQL\_ROOT\_PASSWORD=${MYSQL\_ROOT\_PASSWORD}

- MYSQL\_DATABASE=tutorial\_platform

volumes:

- mysql\_data:/var/lib/mysql

restart: unless-stopped

redis:

image: redis:7-alpine

command: redis-server --requirepass ${REDIS\_PASSWORD}

restart: unless-stopped

volumes:

mysql\_data:

### **4.2 Kubernetes 部署**

#### **4.2.1 部署配置**

yaml

apiVersion: apps/v1kind: Deploymentmetadata:

name: tutorial-platformspec:

replicas: 3

selector:

matchLabels:

app: tutorial-platform

template:

metadata:

labels:

app: tutorial-platform

spec:

containers:

- name: app

image: tutorial-platform:latest

ports:

- containerPort: 8088

env:

- name: SPRING\_PROFILES\_ACTIVE

value: "prod"

- name: DB\_PASSWORD

valueFrom:

secretKeyRef:

name: db-secret

key: password

resources:

requests:

memory: "512Mi"

cpu: "250m"

limits:

memory: "1Gi"

cpu: "500m"

livenessProbe:

httpGet:

path: /actuator/health

port: 8088

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /actuator/health/readiness

port: 8088

initialDelaySeconds: 5

periodSeconds: 5

### **4.3 部署环境**

#### **4.3.1 环境划分**

plaintext

开发环境 (Development)

├── 用途: 日常开发测试

├── 配置: 单实例，内存数据库

└── 更新: 自动部署develop分支

测试环境 (Testing)

├── 用途: 功能测试，集成测试

├── 配置: 单实例，独立数据库

└── 更新: 手动部署feature分支

预生产环境 (Staging)

├── 用途: 生产前验证

├── 配置: 生产环境配置

└── 更新: 手动部署release分支

生产环境 (Production)

├── 用途: 正式服务

├── 配置: 高可用，负载均衡

└── 更新: 手动部署main分支

## **5. 运维计划**

### **5.1 监控体系**

#### **5.1.1 应用监控**

yaml

监控指标:

- 应用健康状态: /actuator/health

- 系统指标: CPU、内存、磁盘使用率

- 业务指标: 请求量、响应时间、错误率

- JVM指标: 堆内存、GC情况、线程数

#### **5.1.2 监控工具栈**

plaintext

Prometheus # 指标收集

├── Grafana # 指标可视化

├── AlertManager # 告警管理

└── Spring Boot Actuator # 应用指标暴露

#### **5.1.3 Grafana 仪表板配置**

json

{

"dashboard": {

"title": "Tutorial Platform Monitoring",

"panels": [

{

"title": "Request Rate",

"type": "graph",

"targets": [

{

"expr": "rate(http\_requests\_total[5m])"

}

]

},

{

"title": "Response Time",

"type": "graph",

"targets": [

{

"expr": "histogram\_quantile(0.95, rate(http\_request\_duration\_seconds\_bucket[5m]))"

}

]

}

]

}}

### **5.2 日志管理**

#### **5.2.1 日志配置**

xml

<!-- logback-spring.xml --><configuration>

<springProfile name="!prod">

<appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="INFO">

<appender-ref ref="CONSOLE"/>

</root>

</springProfile>

<springProfile name="prod">

<appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">

<file>logs/application.log</file>

<rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">

<fileNamePattern>logs/application-%d{yyyy-MM-dd}.%i.gz</fileNamePattern>

<maxFileSize>100MB</maxFileSize>

<maxHistory>30</maxHistory>

<totalSizeCap>3GB</totalSizeCap>

</rollingPolicy>

<encoder class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">

<providers>

<timestamp/>

<logLevel/>

<loggerName/>

<message/>

<mdc/>

<stackTrace/>

</providers>

</encoder>

</appender>

<root level="INFO">

<appender-ref ref="FILE"/>

</root>

</springProfile></configuration>

#### **5.2.2 日志收集**

yaml

# ELK Stack集成

Filebeat # 日志收集

├── Logstash # 日志处理

├── Elasticsearch # 日志存储和搜索

└── Kibana # 日志可视化

### **5.3 备份策略**

#### **5.3.1 数据库备份**

bash

**#!/bin/bash**# 数据库备份脚本

DB\_NAME="tutorial\_platform"BACKUP\_DIR="/backups/mysql"DATE=$(date +%Y%m%d\_%H%M%S)

# 创建备份

mysqldump -u root -p${DB\_PASSWORD} \

--single-transaction \

--routines \

--triggers \

${DB\_NAME} > ${BACKUP\_DIR}/${DB\_NAME}\_${DATE}.sql

# 压缩备份文件gzip ${BACKUP\_DIR}/${DB\_NAME}\_${DATE}.sql

# 删除7天前的备份find ${BACKUP\_DIR} -name "\*.sql.gz" -mtime +7 -delete

echo "Database backup completed: ${DB\_NAME}\_${DATE}.sql.gz"

#### **5.3.2 文件备份**

bash

**#!/bin/bash**# 应用文件备份脚本

APP\_DIR="/app/upload\_data"BACKUP\_DIR="/backups/files"DATE=$(date +%Y%m%d)

# 创建文件备份tar -czf ${BACKUP\_DIR}/files\_${DATE}.tar.gz -C ${APP\_DIR} .

# 保留30天的备份find ${BACKUP\_DIR} -name "files\_\*.tar.gz" -mtime +30 -delete