SpringCloud Zuul配置详解

**(V1.0)**

**基础技术中心2025年7月**

### **修订记录**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 编号 | 日期 | 描述 | 版本 | 作者 | 审核 | 发布日期 |
| 1 | 2025-07-15 | 建立文档 | 1.0 | 汤王 |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |

**声明**：如无中国建设银行的书面许可，任何人都无权复制或利用本文档内容。

### **目录**

[修订记录 1](#_Toc26988)

[目录 1](#_Toc5880)

[Spring Cloud Netflix Zuul 配置模板 7](#_Toc10472)

[1. 基础配置模板 7](#_Toc27532)

[2. 完整生产配置 7](#_Toc8632)

[3. 配置详解 8](#_Toc3742)

[4. 测试方案 9](#_Toc15512)

[5. 性能优化配置 10](#_Toc11224)

[6. 安全增强配置 11](#_Toc11524)

[7. 监控配置 11](#_Toc19818)

[8. 过滤器开发示例 13](#_Toc3687)

[9. 企业级部署架构 14](#_Toc31505)

[10. 最佳实践建议 14](#_Toc9121)

[Zuul 嵌入式反向代理配置指南 15](#_Toc9719)

[1. 基础配置模板 15](#_Toc5077)

[2. 完整生产配置 15](#_Toc28875)

[3. 高级路由配置 17](#_Toc5797)

[4. 配置详解 18](#_Toc7573)

[5. 测试方案 18](#_Toc21310)

[6. 生产优化配置 20](#_Toc30467)

[7. 安全配置 20](#_Toc4330)

[8. 监控配置 22](#_Toc31042)

[9. 企业级部署架构 23](#_Toc3327)

[10. 最佳实践建议 23](#_Toc24239)

[1. 路由策略 ： 23](#_Toc10586)

[2. 性能优化 ： 24](#_Toc5846)

[3. 安全防护 ： 24](#_Toc32730)

[4. 熔断策略 ： 24](#_Toc8856)

[Zuul HTTP 客户端配置指南 24](#_Toc2666)

[1. 基础配置模板 24](#_Toc26237)

[2. 完整生产配置 25](#_Toc19612)

[3. 客户端类型对比 27](#_Toc19564)

[4. 自定义客户端配置 27](#_Toc4837)

[5. 性能优化配置 29](#_Toc6250)

[6. 测试方案 31](#_Toc15612)

[7. 生产建议配置 32](#_Toc12303)

[8. 客户端切换对比 33](#_Toc29675)

[9. 常见问题排查 34](#_Toc21194)

[10. 企业级配置模板 35](#_Toc29127)

[Zuul 敏感头与Cookie管理配置指南 36](#_Toc28979)

[1. 基础安全配置模板 36](#_Toc28729)

[2. 生产级安全配置 36](#_Toc1015)

[3. 配置详解 37](#_Toc15418)

[4. 测试方案 37](#_Toc1832)

[5. 高级场景配置 38](#_Toc8363)

[6. 安全增强方案 38](#_Toc9455)

[7. 企业级部署架构 39](#_Toc9903)

[8. 最佳实践建议 39](#_Toc13045)

[1. 分层防御策略 ： 39](#_Toc25086)

[2. Cookie管理原则 ： 39](#_Toc4)

[3. 监控配置 ： 39](#_Toc28797)

[9. 常见问题排查 40](#_Toc7079)

[10. 动态配置示例 40](#_Toc20443)

[Zuul 忽略头配置指南 40](#_Toc20437)

[1. 基础配置模板 40](#_Toc281)

[2. 生产级配置示例 41](#_Toc1257)

[3. 配置详解 42](#_Toc28945)

[4. Spring Security默认安全头 42](#_Toc19068)

[5. 测试方案 42](#_Toc5761)

[6. 高级配置示例 43](#_Toc21461)

[7. 企业级安全配置 43](#_Toc31011)

[8. 最佳实践建议 44](#_Toc27913)

[1. 分层控制 ： 44](#_Toc13018)

[2. 安全头策略 ： 44](#_Toc22026)

[3. 监控配置 ： 44](#_Toc17154)

[9. 常见问题排查 44](#_Toc6960)

[10. 动态头管理方案 45](#_Toc3415)

[Zuul 管理端点配置指南 45](#_Toc32511)

[1. 基础端点配置 45](#_Toc26907)

[2. 生产安全配置 46](#_Toc26430)

[3. 路由端点详解 47](#_Toc2654)

[5 .监控集成配置 47](#_Toc3991)

[6. 测试方案 48](#_Toc7828)

[7. 企业级配置模板 48](#_Toc5713)

[8. 端点安全矩阵 49](#_Toc32486)

[9. 最佳实践建议 49](#_Toc23541)

[1. 端点保护 ： 49](#_Toc11731)

[2. 审计日志 ： 49](#_Toc13513)

[3. 动态刷新控制 ： 49](#_Toc17188)

[4. 监控告警 ： 49](#_Toc9105)

[10. 故障排查命令 50](#_Toc16449)

[Zuul 迁移模式与本地转发配置指南 50](#_Toc20218)

[1. 基础迁移配置模板 50](#_Toc22630)

[2. 生产级迁移方案 51](#_Toc12171)

[3. 配置策略详解 52](#_Toc21325)

[4. 监控与测试方案 52](#_Toc8739)

[5. 企业级迁移架构 53](#_Toc9791)

[6. 高级配置示例 53](#_Toc11072)

[7. 最佳实践建议 53](#_Toc32435)

[1. 路由优先级控制 ： 53](#_Toc22948)

[2. 流量标记 ： 53](#_Toc4836)

[3. 迁移监控面板 ： 54](#_Toc27420)

[8. 故障排查方案 54](#_Toc28670)

[9. 动态迁移控制 54](#_Toc11371)

[10. 安全注意事项 54](#_Toc31446)

[Zuul 文件上传配置指南 55](#_Toc5551)

[1. 基础文件上传配置 55](#_Toc16203)

[2. 生产级文件上传方案 56](#_Toc28336)

[3. 文件上传方式对比 56](#_Toc19393)

[4. 客户端上传示例 57](#_Toc25346)

[5. 安全配置方案 57](#_Toc337)

[6. 监控与调试 57](#_Toc27618)

[7. 企业级架构方案 58](#_Toc18084)

[8. 故障排查命令 58](#_Toc26155)

[9. 最佳实践建议 58](#_Toc15006)

[1. 分块传输优化 ： 58](#_Toc7393)

[2. 断点续传支持 ： 58](#_Toc11571)

[3. 存储策略分离 ： 59](#_Toc5407)

[10. 安全防护措施 59](#_Toc6179)

[Zuul 查询字符串编码配置指南 59](#_Toc3693)

[1. 基础配置模板 59](#_Toc24186)

[2. 生产级配置方案 60](#_Toc712)

[3. 编码模式对比 61](#_Toc4339)

[4. 测试方案 61](#_Toc3039)

[5. 高级配置示例 61](#_Toc9922)

[6. 监控与调试 62](#_Toc32599)

[7. 企业级解决方案 62](#_Toc21832)

[8. 故障排查命令 62](#_Toc25110)

[9. 最佳实践建议 63](#_Toc18104)

[1. 路由级控制 ： 63](#_Toc4828)

[2. 参数白名单 ： 63](#_Toc11975)

[3. 监控关键指标 ： 63](#_Toc20149)

[10. 安全防护措施 63](#_Toc30156)

[Zuul 请求URI编码配置指南 64](#_Toc1555)

[1. 基础配置模板 64](#_Toc13829)

[2. 生产级配置方案 64](#_Toc28556)

[3. 编码模式对比 65](#_Toc25560)

[4. 测试方案 65](#_Toc28033)

[5. 高级路由配置 65](#_Toc329)

[6. 监控与调试 66](#_Toc18003)

[7. 企业级解决方案 66](#_Toc32258)

[8. 故障排查命令 66](#_Toc27328)

[9. 最佳实践建议 67](#_Toc12436)

[1. 路由级策略 ： 67](#_Toc9374)

[2. 安全校验 ： 67](#_Toc31459)

[3. 监控告警 ： 67](#_Toc6168)

[10. 安全防护措施 67](#_Toc31454)

[Zuul 原生嵌入式模式配置指南 69](#_Toc5133)

[1. 基础配置模板 69](#_Toc29161)

[2. 完整生产配置 70](#_Toc21071)

[3. 核心配置说明 70](#_Toc8355)

[4. 自定义过滤器示例 70](#_Toc3779)

[5. 测试方案 71](#_Toc505)

[6. 企业级架构方案 71](#_Toc21697)

[7. 性能优化配置 71](#_Toc26526)

[8. 监控配置 71](#_Toc7194)

[9. 安全增强方案 72](#_Toc19289)

[10. 最佳实践建议 72](#_Toc23475)

[1. 路由分层 ： 72](#_Toc981)

[2. 过滤器编排 ： 72](#_Toc9621)

[3. 动态路由 ： 73](#_Toc16590)

[Zuul 过滤器禁用配置指南 73](#_Toc16612)

[1. 基础禁用配置模板 73](#_Toc4565)

[2. 完整过滤器控制方案 73](#_Toc6167)

[3. 内置过滤器清单 75](#_Toc5121)

[4. 生产环境推荐配置 75](#_Toc18021)

[5. 测试方案 76](#_Toc1599)

[6. 企业级管理方案 76](#_Toc9554)

[7. 监控配置 76](#_Toc16679)

[8. 安全注意事项 77](#_Toc939)

[9. 最佳实践建议 77](#_Toc13955)

[1. 分层控制 ： 77](#_Toc14721)

[2. 自动化检查 ： 77](#_Toc32225)

[3. 文档化策略 ： 77](#_Toc10467)

[10. 故障排查命令 77](#_Toc24300)

[Zuul Hystrix 熔断降级配置指南 78](#_Toc31066)

[1. 基础降级配置模板 78](#_Toc3937)

[2. 完整降级方案 79](#_Toc3205)

[3. 全局降级配置 80](#_Toc18909)

[4. 生产级配置建议 80](#_Toc23614)

[5. 降级测试方案 81](#_Toc19407)

[6. 监控配置 81](#_Toc14124)

[7. 企业级架构方案 81](#_Toc31834)

[8. 最佳实践建议 82](#_Toc14004)

[1. 分级降级 ： 82](#_Toc19179)

[2. 动态降级 ： 82](#_Toc24997)

[3. 降级内容管理 ： 82](#_Toc3017)

[9. 安全防护措施 82](#_Toc18709)

[10. 故障排查命令 82](#_Toc2547)

[Zuul 超时配置指南 83](#_Toc6469)

[1. 基础超时配置模板 83](#_Toc250)

[2. 生产级超时方案 83](#_Toc9865)

[3. 超时配置矩阵 84](#_Toc1735)

[4. 动态超时配置 84](#_Toc32384)

[5. 测试方案 85](#_Toc27750)

[6. 监控配置 85](#_Toc2097)

[7. 企业级最佳实践 85](#_Toc5707)

[1. 分层超时策略 ： 85](#_Toc20920)

[2. 服务画像 ： 86](#_Toc4020)

[3. 异常处理 ： 86](#_Toc24742)

[8. 故障排查命令 86](#_Toc31245)

[9. 安全配置 86](#_Toc16979)

[10. 多环境配置 86](#_Toc16514)

[Zuul Location Header 重写配置指南 87](#_Toc17315)

[1. 基础配置模板 87](#_Toc31016)

[2. 生产级安全配置 88](#_Toc13328)

[3. 高级控制方案 88](#_Toc26721)

[4. 配置详解 89](#_Toc30206)

[5. 测试方案 89](#_Toc104)

[6. 监控与安全 89](#_Toc28315)

[7. 企业级架构建议 89](#_Toc17953)

[8. 最佳实践 90](#_Toc14813)

[1. 白名单控制 ： 90](#_Toc3502)

[2. 动态配置 ： 90](#_Toc27487)

[3. 审计日志 ： 90](#_Toc21584)

[9. 故障排查 90](#_Toc11046)

[10. 紧急控制方案 90](#_Toc27286)

[Zuul 跨域请求(CORS)配置指南 91](#_Toc13723)

[1. 基础CORS配置模板 91](#_Toc30973)

[2. 生产级CORS方案 92](#_Toc16439)

[3. 多环境CORS配置 92](#_Toc31320)

[4. 动态CORS控制 93](#_Toc4280)

[5. 安全增强配置 93](#_Toc25281)

[6. 测试方案 94](#_Toc17112)

[7. 监控配置 94](#_Toc7916)

[8. 企业级架构建议 94](#_Toc22746)

[9. 最佳实践 95](#_Toc14081)

[1. 分层控制 ： 95](#_Toc21869)

[2. 安全审计 ： 95](#_Toc31090)

[3. 动态更新 ： 95](#_Toc20917)

[10. 故障排查 95](#_Toc30061)

[Zuul 网关监控指标配置指南 96](#_Toc22641)

[1. 基础监控配置 96](#_Toc2512)

[2. 关键指标说明 96](#_Toc22434)

[3. 生产级监控方案 96](#_Toc13516)

[4. 自定义指标扩展 97](#_Toc17439)

[5. 告警规则示例 97](#_Toc29085)

[6. 监控端点测试 98](#_Toc9969)

[7. Grafana仪表板配置 98](#_Toc7420)

[8. 企业级监控架构 98](#_Toc19058)

[9. 最佳实践建议 98](#_Toc24126)

[1. 分层监控 ： 98](#_Toc11649)

[2. 动态采样 ： 99](#_Toc16948)

[3. 指标聚合 ： 99](#_Toc21840)

[10. 故障排查工具 99](#_Toc24142)

[Zuul 深度开发指南 99](#_Toc6799)

[1. 核心架构配置 99](#_Toc17805)

[2. 过滤器开发模板 100](#_Toc13570)

[3. 路由过滤器对比 101](#_Toc9796)

[4. 生产级路由配置 101](#_Toc8396)

[5. 异常处理方案 103](#_Toc24257)

[6. 性能优化配置 103](#_Toc4840)

[7. 监控集成方案 103](#_Toc2127)

[8. 安全增强措施 104](#_Toc7858)

[9. 动态配置示例 104](#_Toc13106)

[10. 调试与测试工具 104](#_Toc31851)

[11. 请求上下文最佳实践 105](#_Toc9093)

[12. 版本迁移策略 105](#_Toc8471)

[13. 性能关键指标 105](#_Toc7726)

# Spring Cloud Netflix Zuul 配置模板

**1.** 基础配置模板

zuul:

*# 网关基础配置*

enabled: true *# 启用Zuul网关功能*

*# 路由规则配置*

routes:

user-service:

path: /api/users/\*\* *# 用户服务路由路径*

serviceId: user-service *# 注册中心服务名*

stripPrefix: true *# 移除路径前缀*

product-service:

path: /api/products/\*\* *# 商品服务路由路径*

url: http://product-service:8080 *# 直接服务地址*

stripPrefix: false *# 保留完整路径*

*# 连接超时设置*

host:

connect-timeout-millis: 2000 *# 连接建立超时(毫秒)*

socket-timeout-millis: 10000 *# 请求响应超时(毫秒)*

*# 安全相关配置*

sensitive-headers: Cookie,Authorization *# 过滤的敏感请求头*

add-proxy-headers: true *# 启用代理头信息*

配置说明：

1. 路由配置支持服务发现和直接URL两种方式
2. 超时设置控制网关与后端服务的交互
3. 安全配置保护敏感信息和代理转发

**2.** 完整生产配置

zuul:

*# 全局配置*

prefix: /v1 *# 统一API前缀*

strip-prefix: true *# 自动剥离前缀*

retryable: true *# 允许请求重试*

*# 路由规则配置*

routes:

auth-service:

path: /auth/\*\* *# 认证服务路径*

serviceId: auth-service *# 服务注册名称*

sensitiveHeaders: *# 特殊头处理*

stripPrefix: false *# 保留完整路径*

inventory-service:

path: /inventory/\*\* *# 库存服务路径*

url: http://inventory-service:8080 *# 直接URL路由*

custom-metadata: *# 自定义元数据*

timeout: 5000 *# 超时时间(毫秒)*

max-retries: 3 *# 最大重试次数*

payment-gateway:

path: /payments/\*\* *# 支付网关路径*

serviceId: payment-service *# 支付服务名称*

filters: *# 过滤器配置*

- pre: 1 *# 前置过滤器(优先级1)*

- post: -1 *# 后置过滤器(优先级-1)*

*# 熔断保护配置*

hystrix:

command:

default: *# 默认配置*

execution:

isolation:

thread:

timeoutInMilliseconds: 10000 *# 默认超时10秒*

payment-service: *# 支付服务专用配置*

execution:

isolation:

thread:

timeoutInMilliseconds: 30000 *# 支付超时30秒*

*# 跨域访问配置*

cors:

allowed-origins: "https://yourdomain.com" *# 允许的域名*

allowed-methods: GET,POST,PUT *# 允许的HTTP方法*

allowed-headers: Content-Type,Authorization *# 允许的请求头*

max-age: 3600 *# 预检请求缓存时间(秒)*

配置说明：

1. 全局配置设置API前缀和重试策略
2. 路由配置定义各服务的访问规则
3. 熔断配置保护服务稳定性
4. 跨域配置控制浏览器访问权限

**3.** 配置详解

| ****配置项**​** | **​**说明**​** | **​**推荐值**​** |
| --- | --- | --- |
| routes.{id}.path | Ant风格路径匹配 | /api/\*\* |
| routes.{id}.serviceId | Eureka服务名 | user-service |
| routes.{id}.url | 静态服务地址 | http://host:port |
| host.connect-timeout-millis | TCP连接超时 | 2000-5000ms |
| host.socket-timeout-millis | 响应读取超时 | 10000-30000ms |
| sensitive-headers | 不过滤的头信息 | Cookie,Authorization |
| hystrix.command.timeout | 熔断超时时间 | 按服务调整 |

**4.** 测试方案

*# ==============================================*

*# Zuul网关测试命令集（带详细注释）*

*# ==============================================*

*# 1. 基础路由功能测试# 测试用户服务路由是否正常转发请求*

*# 请求格式：网关地址 + 全局前缀 + 路由路径 + 参数*

*# 预期结果：正常返回用户ID为123的信息*

curl -v <http://zuul:8080/v1/api/users/123>

*# 2. 网关超时机制验证测试*

*# 通过特殊请求头模拟后端服务延迟（15秒）*

*# 配置中超时设置为10秒，预期会触发网关超时*

*# 预期结果：返回504网关超时错误*

curl -H "X-Simulate-Delay: 15000" <http://zuul:8080/v1/api/products>

*# 3. 熔断降级功能压测*

*# 连续快速请求支付接口，模拟高并发场景*

*# 当失败率达到阈值时应当触发熔断机制*

*# 注意：测试时建议添加间隔，避免过度消耗资源*

while true; do

curl http://zuul:8080/v1/payments

sleep 0.5 *# 每次请求间隔0.5秒*done

*# 4. 跨域访问功能测试*

*# 发送OPTIONS预检请求模拟跨域访问*

*# 测试网关是否正确处理CORS跨域配置*

*# 预期结果：返回包含允许的跨域头信息*

curl -X OPTIONS \

-H "Origin: http://test.com" \

-H "Access-Control-Request-Method: POST" \

http://zuul:8080 -v

*# 5. 动态路由配置查看*

*# 通过Actuator端点获取当前路由配置信息*

*# 需要确保actuator/routes端点已启用*

*# 预期结果：返回JSON格式的路由配置数据*

curl <http://zuul:8080/actuator/routes>

*# 6. （扩展测试）网关健康状态检查# 验证网关基础健康状态# 预期结果：返回{"status":"UP"}表示正常*

curl <http://zuul:8080/actuator/health>

*# 7. （扩展测试）动态配置刷新*

*# 强制刷新路由配置（需配置相应权限）*

*# 适用场景：动态修改配置后立即生效*

*# 注意：生产环境慎用*

curl -X POST http://zuul:8080/actuator/refresh

*# ==============================================*

*# 测试注意事项：*

*# 1. 确保测试前网关服务正常启动*

*# 2. 建议配合日志监控进行测试*

*# 3. 生产环境测试要注意流量控制*

*# 4. 熔断测试后需等待服务恢复# ==============================================*

**5.** 性能优化配置

*# Zuul网关核心配置*

zuul:

host:

*# 连接池配置*

max-total-connections: 200 *# 网关最大总连接数*

max-per-route-connections: 20 *# 单服务最大连接数*

server:

tomcat:

*# 线程池配置*

max-threads: 250 *# 最大工作线程数*

min-spare-threads: 20 *# 最小空闲线程数*

accept-count: 100 *# 请求等待队列长度*

ribbon:

*# 服务调用配置*

ConnectTimeout: 3000 *# 连接超时(毫秒)*

ReadTimeout: 10000 *# 读取超时(毫秒)*

MaxAutoRetries: 1 *# 失败自动重试次数*

配置说明：

1. 连接池：控制网关与后端服务的连接资源
2. 线程池：影响网关自身请求处理能力
3. Ribbon：决定服务调用的超时和重试策略
4. 安全增强配置

zuul:

*# 安全防护配置*

*# IP访问控制*

ignored-patterns:

- /admin/\*\* *# 禁止访问的管理接口路径*

*# 安全路由配置*

routes:

secure-api:

path: /secure/\*\* *# 安全接口路径*

serviceId: secure-service *# 后端服务名*

custom-filter: *# 自定义过滤器*

- name: IPWhitelist *# IP白名单过滤器*

args:

allowed-ips: 192.168.1.0/24 *# 允许的IP段*

*# 敏感头过滤*

sensitive-headers: *# 过滤的敏感请求头*

- Cookie

- Set-Cookie

- Authorization

- X-Forwarded-For

配置说明：

1. 通过ignored-patterns禁用敏感接口
2. 使用自定义过滤器实现IP白名单控制
3. sensitive-headers过滤敏感头信息防止泄露

**7.** 监控配置

management:

endpoints:

web:

exposure:

*# 开放的监控端点*

include: health,info,routes,filters,hystrix.stream

endpoint:

health:

*# 显示健康检查详情*

show-details: always

hystrix:

stream:

*# 启用熔断监控流*

enabled: true

*# 日志级别配置*logging:

level:

*# Zuul核心日志级别*

com.netflix.zuul: DEBUG

*# Spring Cloud Zuul日志级别*

org.springframework.cloud.netflix.zuul: INFO

配置说明：

1. 监控端点配置开放健康检查、路由信息等监控接口
2. 健康检查显示详细信息
3. 启用Hystrix熔断监控流
4. 设置Zuul相关组件的日志级别

**8.** 过滤器开发示例

*// IP白名单过滤器实现*

public class IPWhitelistFilter extends ZuulFilter {

@Override

public String filterType() {

return "pre"; *// 过滤器类型：前置过滤器*

}

@Override

public int filterOrder() {

return 1; *// 执行顺序：优先级1*

}

@Override

public boolean shouldFilter() {

return true; *// 始终启用该过滤器*

}

@Override

public Object run() {

RequestContext ctx = RequestContext.getCurrentContext();

String ip = ctx.getRequest().getRemoteAddr(); *// 获取客户端IP*

if (!isAllowed(ip)) {

*// IP不在白名单时的处理*

ctx.setResponseStatusCode(403); *// 403禁止访问*

ctx.setResponseBody("Forbidden"); *// 返回提示信息*

ctx.setSendZuulResponse(false); *// 不转发请求*

}

return null;

}

*// IP白名单校验方法*

private boolean isAllowed(String ip) {

*// 实现IP白名单校验逻辑*

return false;

}

}

代码说明：

1. 继承ZuulFilter实现自定义过滤器
2. filterType定义过滤器类型为前置过滤
3. filterOrder设置执行优先级
4. run方法包含核心过滤逻辑
5. 拦截非法IP返回403状态码
6. 实际使用时需完善isAllowed方法实现

**9.** 企业级部署架构

[客户端] → [负载均衡器] → [Zuul集群] → [Eureka注册中心] → [微服务]

↓ ↓

[监控系统] ← [配置中心] [熔断仪表盘]

**10.** 最佳实践建议

1. 路由策略 ：

 按业务域划分路由

 为关键服务设置独立超时 . 禁用 ignored-services: '\*'

1. 性能优化 ：

zuul:

*# 连接池配置*

host:

max-per-route-connections: 50 *# 每个路由的最大连接数*

ribbon:

*# Ribbon配置*

eager-load:

enabled: true *# 服务启动时预加载Ribbon配置*

配置说明：

1. max-per-route-connections限制单个服务的最大连接数
2. eager-load在启动时加载Ribbon配置，避免首次请求延迟

3. 安全防护 ：

 启用请求签名验证

 配置严格的CORS策略

 定期审计路由规则 4. 监控告警 ：

 监控P99响应时间  设置熔断阈值告警  记录所有拒绝请求



# Zuul 嵌入式反向代理配置指南

1. 基础配置模板

zuul:

*# 基础配置*

enabled: true *# 启用Zuul代理功能*

*# 路由规则配置*

routes:

users:

path: /myusers/\*\* *# 路由匹配路径*

serviceId: users\_service *# 对应的服务名称*

stripPrefix: false *# 保留完整请求路径*

*# 全局代理配置*

ignoredServices: '\*' *# 忽略所有未配置的服务（安全防护）*

prefix: /api *# 统一添加API前缀*

stripPrefix: true *# 自动移除全局前缀*

addProxyHeaders: true *# 添加X-Forwarded等代理头*

配置说明：

1. 路由配置实现/myusers到users\_service的映射
2. ignoredServices提升安全性，避免暴露未配置服务
3. prefix/stripPrefix组合实现统一路径处理
4. 代理头确保后端获取真实请求信息

**2.** 完整生产配置

zuul:

*# 路由规则配置*

routes:

user-service-v1:

path: /v1/users/\*\* *# 用户服务V1版路径*

serviceId: user-service *# 注册中心服务名*

stripPrefix: true *# 移除路径前缀*

payment-service:

path: /payments/\*\* *# 支付服务路径*

url: https://payment-api.com *# 直接指定服务地址*

stripPrefix: false *# 保留完整路径*

legacy-system:

path: /legacy/\*\* *# 旧系统路径*

serviceId: legacy-service *# 旧系统服务名*

stripPrefix: true *# 移除路径前缀*

retryable: true *# 允许请求重试*

*# 全局通用配置*

ignoredServices: '\*' *# 忽略未显式配置的服务*

prefix: /gateway *# 统一网关前缀*

stripPrefix: true *# 移除全局前缀*

sensitiveHeaders: Cookie,Authorization *# 过滤敏感头*

addProxyHeaders: true *# 添加代理头信息*

*# 熔断保护配置*

hystrix:

command:

default: *# 默认熔断设置*

execution:

isolation:

thread:

timeoutInMilliseconds: 10000 *# 默认10秒超时*

legacy-service: *# 旧系统专用设置*

execution:

isolation:

thread:

timeoutInMilliseconds: 30000 *# 旧系统30秒超时*

*# Ribbon客户端配置*ribbon:

eureka:

enabled: false *# 禁用Eureka服务发现*

*# 旧系统服务配置*

legacy-service:

ribbon:

listOfServers: server1.example.com,server2.example.com *# 手动指定服务实例*

ConnectTimeout: 2000 *# 2秒连接超时*

ReadTimeout: 5000 *# 5秒读取超时*

配置说明：

1. 支持三种路由方式：服务发现、直接URL和手动配置实例
2. 全局配置确保安全性和统一路径处理
3. 熔断设置区分常规服务和旧系统
4. Ribbon配置实现灵活的服务实例管理

**3.** 高级路由配置

zuul:

routes:

*# 正则匹配路由*

regex-mapped:

path: /services/\*\* *# 匹配/services开头的路径*

serviceId: regex-service *# 转发到指定服务*

*# 版本化路由配置*

user-service-v2:

path: /v2/users/\*\* *# 用户服务V2版路径*

serviceId: user-service-v2 *# V2版服务名*

stripPrefix: true *# 移除版本前缀*

*# 外部API路由*

external-api:

path: /external/\*\* *# 外部API路径*

url: https://api.external.com/v3 *# 直接指定外部API地址*

stripPrefix: true *# 移除路径前缀*

*# 路径忽略规则*

ignoredPatterns:

- /\*\*/admin/\*\* *# 忽略所有admin路径*

- /\*\*/internal/\*\* *# 忽略所有internal路径*

*# 自定义路由映射器*

patternServiceRouteMapper:

enabled: true *# 启用自定义映射*

servicePattern: "(?<name>^.+)-(?<version>v.+$)" *# 服务名解析正则*

routePattern: "${version}/${name}" *# 路由路径模板*

配置说明：

1. 支持多种路由方式：正则匹配、版本控制、外部API直连
2. 通过ignoredPatterns屏蔽敏感路径
3. 自定义映射器实现服务名到路径的自动转换
4. 路径处理灵活，支持前缀移除和保留

**4.** 配置详解

| **​配置项​** | **​说明​** | **​示例值​** |
| --- | --- | --- |
| routes.{id}.path | Ant风格路径匹配 | /api/\*\* |
| routes.{id}.serviceId | 服务发现ID | user-service |
| routes.{id}.url | 静态服务地址 | http://service:8080 |
| routes.{id}.stripPrefix | 是否移除路径前缀 | true/false |
| ignoredServices | 忽略的服务列表 | '\*' 或 ['serviceA','serviceB'] |
| prefix | 全局路径前缀 | /api |
| ignoredPatterns | 忽略的路径模式 | /\*\*/admin/\*\* |
| patternServiceRouteMapper | 服务ID到路径的自动映射 | 正则表达式转换规则 |

**5.** 测试方案

*# ======================*

*# Zuul网关测试命令集*

*# ======================*

*# 1. 基础路由功能测试*

*# 测试用户服务V1接口*

*# 预期：返回用户ID为123的信息*

curl http://zuul:8080/gateway/v1/users/123

*# 2. 外部API路由测试*

*# 测试静态URL路由转发*

*# 预期：转发到外部API并返回数据*

curl http://zuul:8080/gateway/external/data

*# 3. 路径过滤测试*

*# 尝试访问被忽略的管理接口*

*# 预期：返回404禁止访问*

curl http://zuul:8080/gateway/v1/users/admin

*# 4. 熔断机制压测*

*# 连续请求旧系统慢接口*

*# 预期：触发熔断时返回fallback*

for i in {1..50}; do

curl http://zuul:8080/gateway/legacy/slow

sleep 0.2 *# 间隔0.2秒防止过载*

done

*# 5. 路由映射规则测试*

*# 验证自定义路由映射器*

*# 预期：自动转换/v2/user-service为/user-service/v2*

curl http://zuul:8080/gateway/v2/user-service/users

*# ======================*

*# 测试说明：*

*# 1. 需先启动zuul服务和对应后端服务*

*# 2. 熔断测试后需等待1分钟恢复*

*# 3. 生产环境慎用压力测试*

*# ======================*

测试要点说明：

1. 基础路由：验证最基本的路由转发功能
2. 静态路由：测试直接URL转发的稳定性
3. 路径过滤：检查安全规则是否生效
4. 熔断测试：验证系统保护机制
5. 映射测试：确认自定义路由规则解析正确

**6.** 生产优化配置

*# ======================*

*# 微服务调用相关配置*

*# ======================*

ribbon:

*# 客户端调用配置*

ConnectTimeout: 2000 *# 建立连接超时(ms)*

ReadTimeout: 10000 *# 请求响应超时(ms)*

MaxAutoRetries: 1 *# 当前实例重试次数*

MaxAutoRetriesNextServer: 2 *# 切换实例重试次数*

OkToRetryOnAllOperations: false *# 仅GET请求自动重试*

hystrix:

command:

default:

circuitBreaker:

*# 熔断器配置*

requestVolumeThreshold: 20 *# 触发熔断的最小请求数*

errorThresholdPercentage: 50 *# 错误率阈值(%)*

sleepWindowInMilliseconds: 5000 *# 熔断恢复时间(ms)*

zuul:

host:

*# 连接池配置*

max-total-connections: 200 *# 最大连接总数*

max-per-route-connections: 50 *# 单服务最大连接数*

*# ======================*

*# 配置说明：*

*# 1. Ribbon控制服务间调用行为*

*# 2. Hystrix提供熔断保护机制*

*# 3. 连接池影响网关吞吐量*

*# ======================*

核心配置说明：

1. Ribbon超时设置应大于业务平均耗时
2. 熔断触发条件：20请求且50%失败率
3. 连接池大小需根据服务器配置调整
4. 生产环境建议开启请求重试但限制次数

**7.** 安全配置

zuul:

*# 安全头过滤配置*

sensitiveHeaders:

- Cookie *# 过滤Cookie头*

- Set-Cookie *# 过滤Set-Cookie头*

- Authorization *# 过滤认证头*

- X-Secret-Token *# 过滤自定义密钥头*

*# 跨域访问控制*

cors:

allowed-origins: "https://yourdomain.com" *# 允许访问的域名*

allowed-methods: "GET,POST" *# 允许的HTTP方法*

allowed-headers: "Content-Type" *# 允许的请求头*

max-age: 3600 *# 预检请求缓存时间(秒)*

*# 敏感路径屏蔽*

ignoredPatterns:

- /actuator/\*\* *# 屏蔽监控端点*

- /admin/\*\* *# 屏蔽管理接口*

安全配置说明：

1. sensitiveHeaders防止敏感信息透传到下游服务
2. cors精确控制跨域访问权限
3. ignoredPatterns保护监控和管理接口
4. 生产环境建议配合SSL加密使用

**8.** 监控配置

management:

endpoints:

web:

exposure:

include: health,info,routes *# 开放的监控端点*

endpoint:

health:

show-details: always *# 显示健康检查详情*

hystrix:

stream:

enabled: true *# 启用熔断监控流*

logging:

level:

com.netflix.zuul: DEBUG *# Zuul核心调试日志*

org.springframework.cloud.netflix.zuul: INFO *# Spring集成日志*

配置说明：

1. 监控端点开放健康检查、基础信息和路由列表
2. 健康检查显示详细组件状态
3. 启用Hystrix仪表盘数据流
4. 设置Zuul不同模块的日志级别

**9.** 企业级部署架构

[客户端] → [负载均衡器] → [Zuul集群] → [服务发现] → [微服务]

↓ ↓

[监控系统] ← [配置中心] [熔断仪表盘]

**10.** 最佳实践建议

1. 路由策略 ：

. 使用 ignoredServices: '\*' 配合显式路由声明  为不同版本服务创建独立路由

. 使用 stripPrefix 控制路径处理

2. 性能优化 ：

server:

tomcat:

max-threads: 250 *# 最大工作线程数（并发处理能力）*

min-spare-threads: 20 *# 最小空闲线程数（保持常驻线程）*

配置说明：

1. max-threads决定系统最大并发量
2. min-spare-threads提高瞬时请求响应速度
3. 建议值：
   * 开发环境：50-100
   * 生产环境：根据CPU核心数×(100-200)计算

3. 安全防护 ：

. 严格限制 allowed-origins  定期审计路由规则

 启用请求签名验证

4. 熔断策略 ：

hystrix:

command:

critical-service: *# 关键服务专用配置*

execution:

isolation:

thread:

timeoutInMilliseconds: 30000 *# 30秒超时阈值*

circuitBreaker:

errorThresholdPercentage: 40 *# 错误率40%触发熔断*

配置说明：

1. 针对critical-service服务的特殊配置
2. 超时时间延长至30秒（适合耗时操作）
3. 错误率达到40%时触发熔断保护
4. 生产环境建议配合监控系统使用

# **Zuul <HTTP>** 客户端配置指南

**1.** 基础配置模板

*# application.yml*

zuul:

host:

*# 连接池配置*

max-total-connections: 200 *# 最大总连接数*

max-per-route-connections: 50 *# 单服务最大连接数*

time-to-live: 60000 *# 连接最大存活时间(毫秒)*

ribbon:

*# HTTP客户端配置*

http:

client:

enabled: true *# 使用Apache HTTP客户端(默认)*

*# 其他客户端配置(可选)*

restclient:

enabled: false *# 禁用RestClient(已废弃)*

okhttp:

enabled: false *# 禁用OKHttp客户端*

配置说明：

1. 连接池设置控制网关并发能力
2. 存活时间避免连接长时间占用
3. 默认启用Apache HTTP客户端
4. 明确禁用不推荐的客户端实现

**2.** 完整生产配置

zuul:

host:

*# 连接池核心配置*

max-total-connections: 500 *# 网关最大总连接数*

max-per-route-connections: 100 *# 单服务最大连接数*

time-to-live: 300000 *# 连接最大存活时间(ms)*

socket-timeout-millis: 10000 *# 响应超时时间(10秒)*

connect-timeout-millis: 2000 *# 连接建立超时(2秒)*

connection-request-timeout-millis: 500 *# 获取连接超时(500ms)*

ribbon:

*# HTTP客户端选择*

http:

client:

enabled: true *# 强制使用Apache HTTP客户端*

okhttp:

enabled: false *# 禁用OKHttp客户端*

restclient:

enabled: false *# 禁用RestClient*

*# Apache客户端高级配置*

httpclient:

max-route-connections: 50 *# 单路由最大连接数*

validate-after-inactivity: 5000 *# 连接空闲验证间隔(5秒)*

evict-idle-connections: true *# 启用空闲连接回收*

eviction-period: 30000 *# 连接回收周期(30秒)*

配置说明：

1. 连接池参数根据服务器配置调整
2. 超时设置需大于业务平均耗时
3. 强制使用Apache客户端保证稳定性
4. 空闲连接回收防止资源泄漏
5. 生产环境建议配合监控调整参数

**3.** 客户端类型对比

| **​客户端类型​** | **​启用方式​** | **​适用场景​** |
| --- | --- | --- |
| Apache HTTP Client | ribbon.http.client.enabled=true | 默认选择，生产推荐 |
| OKHttp | ribbon.okhttp.enabled=true | 需要HTTP/2支持的场景 |
| Ribbon RestClient | ribbon.restclient.enabled=true | 兼容旧系统（已废弃） |

**4.** 自定义客户端配置

*// Apache HTTP客户端自定义配置*

@Bean

public CloseableHttpClient customHttpClient() {

return HttpClientBuilder.create()

.setMaxConnTotal(500) *// 最大连接总数*

.setMaxConnPerRoute(100) *// 单路由最大连接数*

.setConnectionTimeToLive(5, TimeUnit.MINUTES) *// 连接存活时间5分钟*

.evictIdleConnections(30, TimeUnit.SECONDS) *// 30秒回收空闲连接*

.build();

}

*// OKHttp客户端自定义配置（需通过配置启用）*

@Bean

@ConditionalOnProperty("ribbon.okhttp.enabled") *// 根据配置决定是否创建*

public OkHttpClient okHttpClient() {

return new OkHttpClient.Builder()

.connectTimeout(2, TimeUnit.SECONDS) *// 连接超时2秒*

.readTimeout(10, TimeUnit.SECONDS) *// 读取超时10秒*

.connectionPool(new ConnectionPool(

200, *// 最大空闲连接数*

5, *// 保持时间*

TimeUnit.MINUTES))

.build();

}

配置说明：

1.Apache客户端配置：

* + 控制连接池大小和存活时间
  + 自动回收空闲连接节省资源
  + 适合高并发场景

2.OKHttp客户端配置：

* + 需通过ribbon.okhttp.enabled启用
  + 内置连接池管理
  + 超时设置与业务匹配

3.生产建议：

* + 根据压测结果调整连接数
  + 超时时间大于平均响应时间
  + 只启用一种HTTP客户端

**5.** 性能优化配置

*# ======================*

*# HTTP客户端高级调优配置*

*# ======================*

httpclient:

*# 连接池核心配置*

max-total-connections: 1000 *# 最大连接总数*

max-per-route-connections: 100 *# 单路由最大连接数*

time-to-live: 600000 *# 连接存活时间(10分钟)*

*# 连接维护策略*

validate-after-inactivity: 10000 *# 空闲连接验证间隔(10秒)*

evict-idle-connections: true *# 启用空闲连接回收*

eviction-period: 60000 *# 连接清理周期(1分钟)*

*# 超时控制*

socket-timeout: 10000 *# 响应超时(10秒)*

connect-timeout: 2000 *# 连接建立超时(2秒)*

connection-request-timeout: 500 *# 请求连接超时(500ms)*

okhttp:

*# OKHttp专属配置(需手动启用)*

max-idle-connections: 200 *# 最大空闲连接数*

keep-alive-duration: 300000 *# 连接保持时间(5分钟)*

ping-interval: 15000 *# HTTP/2心跳间隔(15秒)*

*# ======================*

*# 调优建议：*

*# 1. 连接数根据QPS和平均响应时间计算*

*# 2. 超时时间应大于业务P99耗时*

*# 3. 生产环境建议开启连接回收# ======================*

配置说明：

1.Apache客户端：

* + 适合每秒500+请求的高并发场景
  + 精确控制连接生命周期
  + 三级超时保障系统稳定性

2.OKHttp客户端：

* + 需配合ribbon.okhttp.enabled=true使用
  + 优化HTTP/2连接复用
  + 内置智能心跳机制

3.通用原则：

* + 连接数 = (QPS × 平均响应时间) / 1000
  + 超时时间 = 业务超时时间 + 缓冲时间(20%)

**6.** 测试方案

*# ======================*

*# HTTP客户端测试命令集*

*# ======================*

*# 1. 连接池压力测试# 模拟100并发，共1000次请求# 观察连接池使用情况和错误率*

ab -n 1000 -c 100 http://zuul:8080/api/test

*# 2. 超时机制验证# 模拟15秒延迟请求（超过10秒超时设置）*

*# 预期结果：触发超时并返回504*

curl -H "X-Simulate-Delay: 15000" http://zuul:8080/api/slow

*# 3. 实时连接池监控# 每秒获取HTTP客户端指标# 关键指标：可用连接数、等待线程数*

watch -n 1 'curl -s http://zuul:8080/actuator/metrics | grep httpclient'

*# 4. 客户端切换测试# 修改配置后执行：*

*# a. 启用OKHttp客户端*echo 'ribbon.okhttp.enabled: true' >> application.yml

*# b. 重启服务后重复测试1-3*

*# c. 对比Apache/OKHttp性能差异*

*# ======================*

*# 测试注意事项：*

*# 1. 压测前确保后端服务就绪*

*# 2. 监控系统资源使用情况*

*# 3. 生产环境建议在隔离环境测试# ======================*

测试要点说明：

1. 连接池测试：验证最大连接数配置是否合理
2. 超时测试：确认超时机制按预期工作
3. 实时监控：观察连接池动态变化
4. 客户端对比：评估不同HTTP客户端性能

**7.** 生产建议配置

*# ======================*

*# 生产环境推荐配置*

*# ======================*

zuul:

host:

*# 连接池计算公式：QPS × 平均响应时间(秒)*

max-total-connections: 800 *# 示例：200QPS × 4秒响应时间*

max-per-route-connections: 100 *# 单服务连接上限*

time-to-live: 300000 *# 连接最大存活时间(5分钟)*

ribbon:

*# HTTP客户端选择*

http:

client:

enabled: true *# 生产推荐Apache客户端*

okhttp:

enabled: false *# 除非需要HTTP/2特性*

*# 监控与指标采集*management:

metrics:

enable:

http:

client: true *# 启用HTTP客户端指标*

distribution:

percentiles:

http.client.requests: 0.95,0.99 *# 采集P95/P99分位数*

*# ======================*

*# 配置说明：*

*# 1. 连接数需根据实际流量计算*

*# 2. Apache客户端稳定性更好*

*# 3. 监控分位数定位性能瓶颈*

*# ======================*

关键配置说明：

1. 连接池大小通过QPS × 平均响应时间动态计算
2. 5分钟连接存活时间平衡性能和资源消耗
3. 监控P95/P99分位数识别慢请求
4. 默认禁用OKHttp除非需要HTTP/2

生产建议：

1. 配合APM工具实时监控
2. 定期根据流量增长调整参数
3. 蓝绿部署验证配置变更

**8.** 客户端切换对比



**9.** 常见问题排查

*# ======================*

*# HTTP客户端运维命令集*

*# ======================*

*# 1. 检查当前HTTP客户端类型*

*# 返回true表示使用Apache客户端*

*# 返回false表示使用OKHttp客户端*

curl -s http://zuul:8080/actuator/env | grep ribbon.http.client.enabled

*# 2. 实时监控连接泄漏*

*# 重点观察：*

*# - leased：已占用连接数*

*# - available：可用连接数*

*# - pending：等待连接线程数*

watch -n 1 'curl -s http://zuul:8080/actuator/metrics/httpclient.connections'

*# 3. 强制释放所有连接*

*# 紧急处理连接泄漏时使用*

*# 会中断正在处理的请求*

curl -X POST http://zuul:8080/actuator/httptrace/reset

*# ======================*

*# 使用说明：*

*# 1. 连接泄漏时优先重启实例*

*# 2. 强制释放后需检查业务影响*

*# 3. 生产环境慎用强制释放*

*# ======================*

命令说明：

1. 客户端检查：确认当前生效的HTTP客户端实现
2. 连接监控：实时观察连接池健康状态
3. 强制释放：紧急恢复手段（会中断请求）

典型问题处理：

1. 连接泄漏表现：
   * leased持续增长不释放
   * available降为零
2. 解决方案：
   * 检查是否有未关闭的响应
   * 适当减少time-to-live
   * 最终手段重启实例

**10.** 企业级配置模板



# 多环境配置示例 spring :

profiles : production

zuul :

host :

max-total-connections : 1000

max-per-route-connections : 150

---

spring :

profiles : development

zuul :

host :

max-total-connections : 200

max-per-route-connections : 20

# 动态配置示例（需配合配置中心） ribbon :

<http:>

client :

enabled: ${[HTTP\_CLIENT\_TYPE:true} #](HTTP_CLIENT_TYPE:true}#) 通过环境变量切换

最佳实践建议 ：

1. 生产环境始终使用Apache [HTTP Client](HTTPClient2)

[2](HTTPClient2). 根据QPS和响应时间动态调整连接池大小

3. 启用连接池监控和自动清理

4. 为关键服务配置独立的连接池参数

5. 定期检查连接泄漏情况

# **Zuul** 敏感头与**Cookie**管理配置指南

## 1. 基础安全配置模板

zuul :

# 全局敏感头设置（默认值）

sensitiveHeaders : Cookie,Set-Cookie,Authorization

routes :

user-service :

path : /api/users/\*\*

serviceId : user-service

# 继承全局敏感头设置

external-api :

path : /external/\*\*

url: [https://third-party.com](https://third-party.com#)

[#](https://third-party.com#) 覆盖全局设置（严格模式）

sensitiveHeaders : Cookie,Set-Cookie,Authorization,X-Secret-Token

## **2. 生产级安全配置**

zuul :

# 全局安全策略

sensitiveHeaders : Cookie,Set-Cookie,Authorization,X-Auth-Token

addProxyHeaders : true

removeSensitiveHeaders : true

routes :

# 内部服务（宽松策略）

internal-auth :

path : /auth/\*\*

serviceId : auth-service

sensitiveHeaders : # 允许传递所有头

customHeaders :

- X-Internal-Request

# 外部服务（严格策略） payment-gateway :

path : /payments/\*\*

url: <https://payment-processor.com>

sensitiveHeaders : Cookie,Set-Cookie,Authorization,X-\*

stripPrefix : true

# 特殊Cookie处理

session-service :

path : /session/\*\*

serviceId : session-service

sensitiveHeaders : Authorization # 仅允许Cookie传递

retainCookie : SESSION\_ID # 白名单Cookie

**3.** 配置详解

| **​配置项​** | **​说明​** | **​推荐值​** |
| --- | --- | --- |
| sensitiveHeaders | 敏感头黑名单 | Cookie,Set-Cookie,Authorization |
| sensitiveHeaders: [] | 允许传递所有头 | 仅限可信内部服务 |
| retainCookie | Cookie白名单 | 特定Cookie名称 |
| addProxyHeaders | 添加代理头 | true |
| removeSensitiveHeaders | 自动移除敏感头 | true |

**4.** 测试方案

# 1 . 测试Cookie传递（应被过滤）

curl -v -H "Cookie : SESSION=123" -H "Authorization : Bearer xxx" \ <http://zuul:8080/api/users>

# 2 . 测试内部服务头传递

curl -v -H "X-Internal-Request : true" -H "Cookie : INTERNAL=456" \

[http://zuul:8080/auth/login](http://zuul:8080/auth/login#3.)

[# 3.](http://zuul:8080/auth/login#3.) 测试白名单Cookie

curl -v -H "Cookie : SESSION\_ID=789; TEMP=abc" \

[http://zuul:8080/session/check #](http://zuul:8080/session/check#) 应只保留SESSION\_ID

# 4 . 验证头过滤效果

curl -s [http://zuul:8080/actuator/headers | jq](http://zuul:8080/actuator/headers|jq) '.headers.sensitive'

**5.** 高级场景配置

zuul :

# 多级Cookie处理 cookie :

policy :

default : DENY # 默认拒绝所有Cookie rules :

- pattern : " .+\\ .yourdomain\\ .com"

policy : ALLOW

retain : SESSION\_ID,LOCALE

- pattern : " .\*payment .\*"

policy : DENY\_ALL

# 动态头管理

dynamic-headers :

enabled : true

injection :

- name : X-Request-ID

value : "${uuid()}"

- name : X-Forwarded-For

expression : "${request .remoteAddr}"

**6.** 安全增强方案

// 自定义Cookie过滤器

public class CookieFilter extends Zuul Filter {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

// 1 . 清理非法Cookie

[HttpServletRequest req = ctx.getRequest](HttpServletRequestreq=ctx.getRequest)();

Cookie[] cookies = req.getCookies();

// . . .验证逻辑 . . .

// 2 . 添加安全标记

ctx .addZuulRequestHeader("X-Zuul-Cookie-Screened", "true");

return null;

}

}

**7.** 企业级部署架构

[浏览器]

↓ (携带Cookie)

[Zuul网关] → [头信息过滤器] → [路由决策] ↓ (净化后的请求)

[内部服务] ←─┐ [外部服务] ←─┘

**8.** 最佳实践建议

1. 分层防御策略 ：

zuul :

sensitiveHeaders : Cookie,Set-Cookie,\*Auth\*

routes :

trusted :

sensitiveHeaders : # 清空表示信任 untrusted :

sensitiveHeaders : \*

2. **Cookie**管理原则 ：

 为每个服务使用独立Cookie名称

 避免使用 <HttpOnly> 标记的Cookie跨服务传递

 定期审计Cookie使用情况

3. 监控配置 ：

management :

endpoints :

web :

exposure :

include : cookies,headers

metrics :

tags :

cookie : "${request .cookies}"

**9.** 常见问题排查

# 1 . 查看实际传递的请求头

curl -v [http://zuul:8080/actuator/httptrace](http://zuul:8080-v#3.)

[# 2.](http://zuul:8080-v#3.) 测试特定头是否被过滤

curl -H "X-Test-Header: value" [http://zuul:8080 -v](http://zuul:8080-v#3.)

[# 3.](http://zuul:8080-v#3.) 强制覆盖敏感头设置（调试用 ）

curl -H "Zuul-Sensitive-Headers: " <http://zuul:8080>

**10.** 动态配置示例



# 基于环境的头管理 spring :

profiles : production

zuul :

sensitiveHeaders : Cookie,Set-Cookie,Authorization,X-\*

---

spring :

profiles : development

zuul :

sensitiveHeaders : # 开发环境允许所有头

关键安全提示 ：

1. 生产环境必须配置 sensitiveHeaders

2. 跨域服务必须使用不同的Cookie名称

3. 定期使用安全工具扫描头信息泄漏

4. 禁用 X-Forwarded-\* 头自动填充（需手动配置）

5. 配合WAF实施额外防护

# Zuul 忽略头配置指南

**1.** 基础配置模板

zuul :

# 全局忽略头设置 ignoredHeaders :

- X-Security-Token

- X-Experimental-Feature

# 安全头处理（当Spring Security存在时）

ignoreSecurityHeaders : false # 是否忽略Spring Security的默认安全头

**2.** 生产级配置示例

zuul :

# 全局忽略头配置（适用于所有路由） ignoredHeaders :

- X-Internal-Debug

- X-Test-Mode

- X-Backend-Version

- X-Env-Info

# 保留Spring Security的安全头 ignoreSecurityHeaders : false

# 路由级覆盖配置 routes :

user-service :

path : /users/\*\*

ignoredHeaders : # 覆盖全局配置

- X-User-Test

- X-Debug-Mode

sensitiveHeaders : Cookie

monitoring-service :

path : /monitor/\*\*

ignoredHeaders : [] # 不忽略任何头

**3.** 配置详解

| **​配置项​** | **​说明​** | **​默认值​** |
| --- | --- | --- |
| zuul.ignoredHeaders | 全局忽略头列表 | 空（有Spring Security时为安全头） |
| zuul.ignoreSecurityHeaders | 是否忽略Spring Security的默认安全头 | true |
| routes.{id}.ignoredHeaders | 路由级忽略头配置 | 继承全局配置 |

**4. Spring Security**默认安全头

当存在Spring Security时，默认会忽略以下安全头：

Cache-Control : no-cache, no-store, max-age=0, must-revalidate

Pragma : no-cache

Expires : 0

X-Content-Type-Options : nosniff

X-Frame-Options : DENY

X-XSS-Protection : 1; mode=block

**5.** 测试方案

# 1 . 测试全局忽略头

curl -v -H "X-Security-Token : test" -H "X-Internal-Debug : true" \ <http://zuul:8080/users>

# 2 . 测试路由级覆盖

curl -v -H "X-User-Test: value" [http://zuul:8080/users/123](http://zuul:8080/users/123#3.)

[# 3.](http://zuul:8080/users/123#3.) 验证安全头传递

curl -v -H "Cache-Control: no-store" [http://zuul:8080/monitor](http://zuul:8080/users/123#3.)

[# 4.](http://zuul:8080/users/123#3.) 查看实际传递的头

curl -s [http://zuul:8080/actuator/httptrace | jq](http://zuul:8080/actuator/httptrace|jq) '.headers'

**6.** 高级配置示例

zuul :

# 动态忽略头配置

dynamic IgnoredHeaders :

enabled : true

patterns :

- "X-Test- .\*" # 忽略所有X-Test-开头的头

- " .\*-Debug" # 忽略所有以-Debug结尾的头

# 按环境配置

environmentSpecificHeaders :

production :

ignoredHeaders :

- X-Debug-Info

development :

ignoredHeaders : []

**7.** 企业级安全配置

security :

headers :

# 禁用Spring Security的默认安全头

cache : false

content-type : false

frame-options : false

xss-protection : false

zuul :

# 手动管理安全头 ignoredHeaders :

- X-Experimental

- X-Beta-Feature

ignoreSecurityHeaders : true # 完全接管安全头管理

# 安全头白名单

allowedSecurityHeaders :

- Strict-Transport-Security

- Content-Security-Policy

**8.** 最佳实践建议

1. 分层控制 ：

zuul :

ignoredHeaders : &default-ignored

- X-Temp

- X-Debug routes :

critical-service :

ignoredHeaders : \*default-ignored # 继承全局 test-service :

ignoredHeaders : [] # 不忽略

2. 安全头策略 ：

. 生产环境应保留 X-Content-Type-Options . API网关统一设置 Cache-Control

. 禁用 X-Powered-By 等信息头

3. 监控配置 ：

management :

metrics :

tags :

header : "${header .key}"

endpoint :

headers :

enabled : true

**9.** 常见问题排查

# 1 . 查看当前生效的忽略头配置

curl -s [http://zuul:8080/actuator/env/zuul.ignoredHeaders | jq](http://zuul:8080/actuator/env/zuul.ignoredHeaders|jq#2.)

[# 2.](http://zuul:8080/actuator/env/zuul.ignoredHeaders|jq#2.) 测试特定头是否被忽略

watch -n 1 'curl -H "X-Test-Header: value" [http://zuul:8080 -v 2](http://zuul:8080-v2)>&1 | grep X-Test-Heade

r

'

# 3 . 临时禁用所有头过滤（调试用 ）

curl -H "Zuul-Ignore-Headers: false" <http://zuul:8080>

**10.** 动态头管理方案



// 自定义头过滤器

public class DynamicHeaderFilter extends Zuul Filter {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

// 动态添加/移除头

if (ctx .getRequest() .getHeader("X-Special-Case") != null) {

ctx .addZuulRequestHeader("X-Special-Processed", "true");

}

return null;

}

}

关键安全提示 ：

1. 生产环境必须审计 ignored Headers 配置

2. 禁止忽略 Content-Security-Policy 等关键安全头

3. 配合WAF实施额外的头过滤

4. 定期扫描敏感头泄漏

5. 为不同环境设置差异化的头策略

# Zuul 管理端点配置指南

**1.** 基础端点配置

management :

endpoints :

web :

exposure :

include : health,info,routes,filters # 暴露的关键端点 base-path : /internal/admin # 管理端点基础路径

endpoint :

routes :

enabled : true

cache :

time-to-live : 30s # 路由缓存时间

filters :

enabled : true

show-sensitive : false # 隐藏敏感过滤器细节

**2.** 生产安全配置

security :

roles : ADMIN # 端点访问角色

zuul :

ignored-patterns :

- /actuator/\*\* # 需配合security配置使用

management :

endpoints :

web :

cors : # 跨域配置

allowed-origins: "<https://ops.yourdomain.com>"

allowed-methods : GET,HEAD

health :

show-details : when-authorized

metrics :

tags :

endpoint : "${endpoint .id}"

**3.** 路由端点详解



**5 .监控集成配置**

```yaml

management :

metrics :

export :

prometheus :

enabled : true

distribution :

percentiles :

zuul .routes .refresh : 0 .95,0 .99

web :

server :

auto-time-requests : true

**6.** 测试方案

# 1 . 测试路由端点

curl -u admin:password <http://zuul:8080/internal/admin/routes>

curl -X POST [http://zuul:8080/internal/admin/routes](http://zuul:8080/internal/admin/routes#2.)

[# 2.](http://zuul:8080/internal/admin/routes#2.) 测试过滤器端点

curl -s [http://zuul:8080/internal/admin/filters | jq](http://zuul:8080/internal/admin/filters|jq) '.pre[]'

# 3 . 验证端点安全

curl -v [http://zuul:8080/internal/admin/routes #](http://zuul:8080/internal/admin/routes#) 应返回401

# 4 . 性能测试

ab -n 1000 -c 10 -A admin:password <http://zuul:8080/internal/admin/routes>

**7.** 企业级配置模板

spring :

profiles : production

management :

endpoint :

routes :

enabled : true

cache :

time-to-live : 1m # 生产环境延长缓存 filters :

enabled : true

metrics :

tags :

zone : "${eureka .instance .metadata-map.zone}"

---

spring :

profiles : development

management :

endpoint :

routes :

cache :

time-to-live : 5s # 开发环境快速刷新

**8.** 端点安全矩阵



**9.** 最佳实践建议

1. 端点保护 ：

security :

user :

name : actuator-admin

password : ${ACTUATOR\_PASSWORD}

roles : ACTUATOR\_ADMIN

2. 审计日志 ：

@Bean

public FilterRegistrationBean<Request LoggingFilter> loggingFilter() {

FilterRegistrationBean<Request LoggingFilter> registration = new FilterRegistrationB

ean<>();

registration .set Filter(new Request LoggingFilter());

registration .addUrlPatterns("/internal/admin/\*");

return registration;

}

3. 动态刷新控制 ：

# 通过消息总线触发刷新（替代直接POST ）

curl -X POST -H "X-Event : RefreshRoutes" \ <http://message-bus/events>

4. 监控告警 ：

alerts :

zuul :

routes-refresh-failure :

enabled : true

threshold : 3/5m # 5分钟内失败3次触发

filters-modified :

severity : warning

**10.** 故障排查命令



# 1 . 检查端点是否启用

curl -s [http://zuul:8080/actuator | jq](http://zuul:8080/actuator|jq) '.\_links'

# 2 . 查看过滤器加载状态

curl -u admin:password <http://zuul:8080/internal/admin/filters> \

| jq ' .error | select( . != null) '

# 3 . 强制重新加载配置

curl -X POST -u operator:password \

-H "X-Force-Refresh : true" \

[http://zuul:8080/internal/admin/routes](http://zuul:8080/internal/admin/routes#4.)

[# 4.](http://zuul:8080/internal/admin/routes#4.) 监控端点性能

watch -n 1 'curl -s <http://zuul:8080/internal/admin/metrics> \

| grep "zuul .admin .requests" '

生产环境必须 ：

1. 使用<HTTPS>保护管理端点

2. 实现IP白名单限制

3. 配置端点访问审计

4. 禁用 /shutdown 端点

5. 定期轮换认证凭据

# Zuul 迁移模式与本地转发配置指南

**1.** 基础迁移配置模板

zuul :

routes :

# 新服务端点

new-service :

path : /api/v2/\*\*

url: [https://new-service.example.com](https://legacy.example.comstripPrefix:false)

[stripPrefix: true](https://legacy.example.comstripPrefix:false)

[#](https://legacy.example.comstripPrefix:false) 本地转发端点

migrated-endpoint :

path : /api/legacy/\*\*

url : forward :/api/v3 # 转发到本地Controller

stripPrefix : false

# 遗留系统（兜底路由 ）

legacy-system :

path : /\*\*

url: [https://legacy.example.com](https://legacy.example.comstripPrefix:false)

[stripPrefix: false](https://legacy.example.comstripPrefix:false)

**2.** 生产级迁移方案

zuul :

routes :

# 阶段一 ：新功能直接路由 feature-new :

path : /features/\*\*

url: [https://modern-system.com](https://modern-system.commetadata:)

[metadata:](https://modern-system.commetadata:)

stage : phase1

owner : team-ark

# 阶段二 ：兼容层转发

compatibility-layer :

path : /compat/\*\*

url : forward :/modern-adapter

stripPrefix : true

metadata :

stage : phase2

# 遗留系统兜底

legacy-fallback :

path : /\*\*

url: [https://old-system.example.com](https://modern-system.commetadata:)

[sensitiveHeaders: #](https://modern-system.commetadata:) 严格过滤头 metadata :

deprecated : true

**3.** 配置策略详解



**4.** 监控与测试方案

# 1 . 测试路由优先级

curl -v [http://zuul:8080/api/v2/features #](http://zuul:8080/api/v2/features#) 应路由到新服务

curl -v [http://zuul:8080/old-api #](http://zuul:8080/old-api#) 应走遗留系统

# 2 . 验证本地转发

curl -v [http://zuul:8080/api/legacy/data #](http://zuul:8080/api/legacy/data#) 应转发到本地/v3端点

# 3 . 监控迁移流量

watch -n 1 'curl -s [http://zuul:8080/actuator/metrics | grep](http://zuul:8080/actuator/metrics|grep) "zuul.route.requests"'

# 4 . 强制刷新路由

curl -X POST <http://zuul:8080/actuator/routes>

**5.** 企业级迁移架构

[客户端] → [Zuul网关] → [新服务集群]

↓

[本地适配层] → [遗留系统]

↓

[监控告警系统]

**6.** 高级配置示例

zuul :

routes :

# 带版本控制的迁移

v1-api :

path : /v1/\*\*

url: [https://legacy-api.example.com](https://legacy-api.example.commetadata:)

[metadata:](https://legacy-api.example.commetadata:)

version : deprecated

v2-api :

path : /v2/\*\*

url : forward :/api/v2-adapter

stripPrefix : true

# 功能开关控制

experimental :

path : /beta/\*\*

url : ${FEATURE\_FLAG:forward :/beta-fallback}

stripPrefix : false

**7.** 最佳实践建议

1. 路由优先级控制 ：

routes :

high-priority :

path : /critical/\*\*

order : -1 # 最高优先级 low-priority :

path : /\*\*

order : 1000

2. 流量标记 ：

// 在过滤器中标记请求来源

RequestContext ctx = RequestContext .getCurrentContext();

ctx .addZuulRequestHeader("X-Route-Source",

ctx .getRequest() .getRequestURI() .startsWith("/v2/") ? "new" : "legacy");

3. 迁移监控面板 ：

management :

metrics :

tags :

route .type : "${route .metadata[ 'stage ']}"

distribution :

percentiles :

zuul .migration .latency : 0 .95,0 .99

**8.** 故障排查方案

# 1 . 查看路由匹配结果

curl -s [http://zuul:8080/actuator/routes/details | jq](http://zuul:8080/actuator/routes/details|jq) '.[] | select(.metadata)'

# 2 . 测试转发链路

curl -v -H "X-Debug-Route: true" [http://zuul:8080/api/transition](http://zuul:8080/api/transition#3.)

[# 3.](http://zuul:8080/api/transition#3.) 强制触发降级

curl -X PUT -d '{"pattern" :"/v1/\*\*","action" :"disable"} ' \

<http://zuul:8080/actuator/route-control>

**9.** 动态迁移控制

@Bean

public Zuul Filter dynamicRouteFilter() {

return new Zuul Filter() {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

if (isInMigrationWindow()) {

ctx .set("serviceId", determineTargetVersion());

}

return null;

}

};

}

**10.** 安全注意事项



security :

migration :

# 迁移期间的特殊安全策略 temporary-policies :

- pattern : /transition/\*\* headers :

- X-Transition-Token methods : GET,POST

- pattern : /v1/\*\*

require-ssl : true

关键迁移原则 ：

1. 始终保持至少一个兜底路由

2. 新老系统共享的接口保持幂等性

3. 为每个迁移阶段建立独立监控

4. 实现自动化回滚机制

5. 在请求头中携带迁移标记（如 X-Migration-Phase ）

**Zuul** 文件上传配置指南

**1.** 基础文件上传配置

zuul :

servlet-path : /zuul # 大文件上传专用路径 routes :

file-service :

path : /files/\*\*

serviceId : file-storage-service

sensitiveHeaders : # 允许上传文件头

stripPrefix : true

# 超时配置（单位 ：毫秒） hystrix :

command :

default :

execution :

isolation :

thread :

timeoutInMilliseconds : 120000 # 大文件超时时间

ribbon :

ConnectTimeout : 5000

# 连接超时

# 读取超时

ReadTimeout : 120000

**2.** 生产级文件上传方案

spring :

servlet :

multipart :

max-file-size : 10MB # 小文件限制

max-request-size : 100MB

zuul :

# 大文件专用配置 large-file :

enabled : true

max-size : 2GB # 大文件限制

buffer-size : 8KB # 缓冲区大小

temp-directory : /tmp/zuul\_uploads

routes :

media-upload :

path : /media/upload/\*\*

serviceId : media-service

custom-sensitive-headers : # 允许文件相关头

- Content-Type

- Content-Length

- Content-Disposition

**3.** 文件上传方式对比



**4.** 客户端上传示例

# 小文件上传（常规路径）

curl -X POST -F "file=@report.pdf" [http://gateway:8080/files/upload](http://gateway:8080/files/upload#)

[#](http://gateway:8080/files/upload#) 大文件上传（专用路径）

curl -v -H "Transfer-Encoding : chunked" \

-H "Content-Type : multipart/form-data" \

-F "file=@4k-video .mp4" \

<http://gateway:8080/zuul/files/upload>

**5.** 安全配置方案

zuul :

# 文件类型白名单

allowed-file-types :

- application/pdf

- image/jpeg

- video/mp4

# 病毒扫描集成 virus-scan :

enabled : true

endpoint: [http://clamav-service/scan](http://gateway:8080/files/upload#)

[timeout: 30000](http://gateway:8080/files/upload#)

**6.** 监控与调试

management :

metrics :

distribution :

percentiles :

zuul .file .upload .size : 0 .95,0 .99

tags :

file .type : "${request .getHeader( 'Content-Type ')}"

endpoints :

file-stats :

enabled : true

cache :

ttl : 10s

**7.** 企业级架构方案

[客户端] → [LB] → [Zuul集群] → [文件存储服务]

↓

[病毒扫描服务] ← [临时存储]

**8.** 故障排查命令

# 1 . 检查文件上传配置

curl -s [http://zuul:8080/actuator/env/zuul.servletPath | jq](http://zuul:8080/actuator/env/zuul.servletPath|jq#2.)

[# 2.](http://zuul:8080/actuator/env/zuul.servletPath|jq#2.) 测试大文件通道

dd if=/dev/zero bs=1M count=1024 | curl -v -H "Transfer-Encoding : chunked" \ -T - [http://zuul:8080/zuul/files/test](http://zuul:8080/actuator/env/zuul.servletPath|jq#2.)

[# 3.](http://zuul:8080/actuator/env/zuul.servletPath|jq#2.) 监控上传流量

watch -n 1 'curl -s <http://zuul:8080/actuator/metrics/zuul.file.upload.size> '

**9.** 最佳实践建议

1. 分块传输优化 ：

zuul :

<httpclient:>

chunked-buffer-size : 16KB # 分块缓冲区大小

max-chunk-size : 512KB # 最大分块大小

2. 断点续传支持 ：

@Bean

public Zuul Filter resumeUpload Filter() {

return new Zuul Filter() {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

if (ctx .getRequest() .getHeader("Range") != null) {

ctx .set("chunkedTransfer", true);

}

return null;

}

};

}

3. 存储策略分离 ：

zuul :

routes :

temp-storage :

path : /upload/temp/\*\*

url : forward :/local-cache # 本地暂存

permanent-storage :

path : /upload/store/\*\*

url: [http://s3-gateway #](http://s3-gateway#) 对象存储

**10.** 安全防护措施



security :

file-upload :

max-files-per-request : 5 # 单次最多文件数

scan-mode : async # 异步病毒扫描

quarantine-folder : /var/quarantine

block-executables : true # 拦截可执行文件

zuul :

filter :

file-type-validator :

enabled : true

allowed-extensions : [ .pdf, .jpg, .mp4]

关键注意事项 ：

1. 生产环境必须限制文件类型

2. 大文件必须使用 /zuul/\* 路径

3. 确保客户端启用分块传输编码

4. 网关层不做文件持久化存储

5. 实现病毒扫描和内容校验

# Zuul 查询字符串编码配置指南

**1.** 基础配置模板

zuul :

# 强制保留原始查询字符串编码

forceOriginalQueryStringEncoding : true # 默认false

**2.** 生产级配置方案

zuul :

query-string :

encoding :

mode : ORIGINAL # 可选值 : ORIGINAL/MODIFIED/STRICT

excluded-routes :

- dynamic-search # 对这些路由禁用原始编码 strict-charset : UTF-8 # 严格模式下的编码格式

routes :

# 需要参数修改的路由 dynamic-search :

path : /search/\*\*

serviceId : search-service

retain-query-params : true # 允许修改参数

# 需要原始编码的路由 legacy-api :

path : /legacy/\*\*

url: [http://old-system.com](http://old-system.comforce-original-encoding:true)

[force-original-encoding: true](http://old-system.comforce-original-encoding:true)

**3.** 编码模式对比



**4.** 测试方案

# 1 . 测试原始编码保留

curl -v "<http://zuul:8080/legacy/api?query=%E4%B8%AD%E6%96%87>" # 中文编码测试

# 2 . 测试参数修改路由

curl -v "<http://zuul:8080/search?q=test&sort=desc>" # 观察参数处理

# 3 . 验证编码一致性

diff <(echo "%E4%B8%AD%E6%96%87" | xxd) <(curl -s "<http://zuul:8080/legacy/decode>" | xx d)

**5.** 高级配置示例

@Bean

public Zuul Filter queryStringFilter() {

return new Zuul Filter() {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

if (ctx .getRequest() .getRequestURI() .contains("/special/")) {

ctx .set("queryStringEncoding", "RFC3986"); // 强制RFC3986编码

}

return null;

}

};

}

**6.** 监控与调试

management :

metrics :

tags :

query.encoding : "${request .queryStringEncoding}"

distribution :

percentiles :

zuul .query .process .time : 0 .95,0 .99

logging :

level :

org.springframework .cloud .netflix .zuul .filters .SimpleHostRoutingFilter : DEBUG

**7.** 企业级解决方案

zuul :

query-string :

decoder :

class : com .company.CustomQueryDecoder # 自定义解码器 encoder :

class : com .company .RFC3986Encoder # 自定义编码器

validation :

enabled : true

rules :

- pattern : '[\w\-% .]+ ' # 允许的字符集

max-length : 1024

**8.** 故障排查命令

# 1 . 查看实际编码效果

curl -vs "<http://zuul:8080/proxy?q=>测试" 2>&1 | grep -E "GET /|Host:"

# 2 . 对比编码差异

diff -u <(curl -s "<http://direct:8081?q=1>+2") <(curl -s "<http://zuul:8080/proxy?q=1>+2")

# 3 . 调试日志输出

tail -f /var/log/zuul/debug.log | grep QueryString

**9.** 最佳实践建议

1. 路由级控制 ：

routes :

strict-api :

path : /strict/\*\*

query-encoding :

mode : STRICT

charset : UTF-8

2. 参数白名单 ：

public class SafeQueryFilter extends Zuul Filter {

@Override

public Object run() {

validateParams(RequestContext .getCurrentContext() .getRequestQueryParams());

}

}

3. 监控关键指标 ：

alerts :

query-string :

encoding-error :

threshold : 5/1m # 每分钟5次编码错误触发告警 malformed-params :

severity : critical

**10.** 安全防护措施

security :

query-string :

max-params : 20

# 最大参数个数

# 最大总长度

max-length : 4096

block-sql-injection : true # SQL注入检测

allowed-special-chars : "-\_ .~" # 允许的特殊字符

关键注意事项 ：

1. forceOriginalQueryStringEncoding 与参数修改功能互斥

2. 包含 + 号等特殊字符的参数需要特别处理

3. 对接支付网关等敏感系统时应启用STRICT模式

4. 需要参数修改的路由应明确禁用原始编码

5. 监控编码不一致导致的后端400错误

# Zuul 请求URI编码配置指南

**1.** 基础配置模板

zuul :

# URI编码处理模式

decodeUrl : false # 默认true（ 自动解码URI ）

routes :

special-path :

path : /special%2Fpath/\*\* # 包含编码字符的路由

url: <http://backend-service>

forceOriginalUri : true # 强制使用原始URI

**2.** 生产级配置方案

zuul :

uri-handling :

mode : ORIGINAL # 可选值 : ORIGINAL/DECODED/STRICT

excluded-paths :

- /api/v2/\*\* # 对这些路径保持自动解码

strict-encoding : UTF-8 # 严格模式下的编码格式

routes :

encoded-api :

path : /encoded/%2Fapi/\*\*

serviceId : legacy-service

uri-handling : ORIGINAL # 路由级覆盖配置

modern-api :

path : /api/v3/\*\*

serviceId : modern-service

uri-handling : DECODED # 显式要求解码

**3.** 编码模式对比



**4.** 测试方案

# 1 . 测试编码URI传递（需decodeUrl=false ）

curl -v "<http://zuul:8080/special%2Fpath/data>" # %2F应被保留

# 2 . 测试自动解码（ decodeUrl=true ）

curl -v "<http://zuul:8080/encoded%2Fapi/value>" # 应自动解码匹配路由

# 3 . 验证严格模式

curl -v "<http://zuul:8080/strict/%E4%B8%AD%E6%96%87>" | jq . # 中文URI测试

**5.** 高级路由配置

routes :

# 动态编码控制路由 smart-encoding :

path : /smart/\*\*

serviceId : adaptive-service

uri-processing :

detect-encoding : true # 自动检测URI编码

fallback : UTF-8

# 双重编码场景

double-encoded :

path : /double/%252F test/\*\* # 双重编码%25 -> %

url: [http://legacy-service](http://legacy-servicedecode-level:2#)

[decode-level: 2 #](http://legacy-servicedecode-level:2#) 需要解码两次

**6.** 监控与调试

management :

metrics :

tags :

uri .encoding : "${request .uriEncoding}"

distribution :

percentiles :

zuul .uri .decode .time : 0 .95,0 .99

logging :

level :

org.springframework .cloud .netflix .zuul .filters .pre .PreDecorationFilter : DEBUG

**7.** 企业级解决方案

@Bean

public Zuul Filter uriEncodingFilter() {

return new Zuul Filter() {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

if (ctx .getRequest() .getRequestURI() .contains("%2F")) {

ctx .put("preserve .encoded .slash", true);

}

return null;

}

};

}

**8.** 故障排查命令

# 1 . 查看实际URI处理结果

curl -vs "<http://zuul:8080/test%2Fpath>" 2>&1 | grep -E "GET|Host:"

# 2 . 对比编码差异

diff -u <(echo -n "/test%2Fpath" | xxd) <(curl -s "<http://zuul:8080/echo-uri>" | xxd)

# 3 . 调试日志输出

tail -f /var/log/zuul/debug.log | grep "URI encoding"

**9.** 最佳实践建议

1. 路由级策略 ：

routes :

legacy :

path : /legacy/\*\*

uri-handling :

mode : ORIGINAL

allow : ["%2F", "%3A"] # 允许的编码字符 modern :

path : /modern/\*\*

uri-handling : DECODED

2. 安全校验 ：

public class UriValidationFilter extends Zuul Filter {

@Override

public Object run() {

String uri = ctx .getRequest() .getRequestURI();

if ( !isValidEncoding(uri)) {

ctx .setResponseStatusCode(400);

}

}

}

3. 监控告警 ：

alerts :

uri-encoding :

malformed-uri :

threshold : 5/1m

severity : high

double-encoding :

pattern : "%25[0-9a-fA-F]{2}"

action : log

**10.** 安全防护措施

security :

uri :

max-length : 2048 # URI最大长度 allowed-schemes: [<http>, <https>] block :

- " . ./"

- "%00"

- "< !--"

# 路径遍历攻击

# 空字节攻击

# XSS攻击

zuul :

filters :

uri-sanitizer :

enabled : true

mode : AGGRESSIVE



关键注意事项 ：

1. decodeUrl=false 时需确保手动设置正确的URI编码

2. 包含 %2F 等编码字符的路由需要特殊处理

3. 启用严格编码模式可能影响URI参数传递

4. 监控URI解码失败的400错误

5. 重要系统应实现URI签名校验

# Zuul 原生嵌入式模式配置指南

**1.** 基础配置模板

# application .yml zuul :

# 纯Zuul服务配置（无服务发现和代理） routes :

api-path :

path : /api/\*\* # 映射到Zuul过滤器链

admin-path :

path : /admin/\*\*

ignored-patterns :

- /internal/\*\* # 排除路径

**2.** 完整生产配置

zuul :

# 路由定义（仅路径映射） routes :

rest-api :

path : /v1/\*\*

custom-metadata :

category : public

internal-api :

path : /internal/v2/\*\*

required-headers : X-Auth-Token

# 全局过滤器设置 filter :

pre :

auth-filter :

order : -100

post :

audit-filter :

order : 1000

# 禁用自动代理功能 ribbon :

enabled : false

eureka :

enabled : false

**3.** 核心配置说明



**4.** 自定义过滤器示例

// 认证过滤器

public class AuthFilter extends Zuul Filter {

@Override

public Object run() {

[HttpServletRequest req = RequestContext.getCurrentContext](HttpServletRequestreq=RequestContext.getCurrentContext)().getRequest();

if (req.getHeader("X-Auth") == null) {

throw new AuthException();

}

return null;

}

}

// 注册过滤器

@Bean

public AuthFilter authFilter() {

return new AuthFilter();

}

**5.** 测试方案

# 1 . 测试路由匹配

curl -v [http://zuul:8080/api/users](http://zuul:8080/internal/health#3.)

[# 2.](http://zuul:8080/internal/health#3.) 测试忽略路径

curl -v [http://zuul:8080/internal/health](http://zuul:8080/internal/health#3.)

[# 3.](http://zuul:8080/internal/health#3.) 验证过滤器链

curl -H "X-Test-Filter: true" <http://zuul:8080/v1/test>

**6.** 企业级架构方案

[客户端] → [嵌入式Zuul] → [自定义过滤器链]

↓

[业务处理层] ← [共享库]

**7.** 性能优化配置

server :

tomcat :

max-threads : 200

accept-count : 100

zuul :

host :

max-per-route-connections : 50

max-total-connections : 500

**8.** 监控配置

management :

endpoints :

web :

exposure :

include : health,metrics,filters

metrics :

tags :

route : "${route .id}"

distribution :

percentiles :

zuul .filter .latency : 0 .95,0 .99

**9.** 安全增强方案

// SQL注入防护过滤器

public class SqlInjectionFilter extends Zuul Filter {

@Override

public Object run() {

// 参数检查逻辑

return null;

}

}

**10.** 最佳实践建议

1. 路由分层 ：

routes :

public-api :

path : /public/\*\*

security-level : low

private-api :

path : /private/\*\*

security-level : high

2. 过滤器编排 ：

@Bean

public FilterRegistrationBean<CorsFilter> corsFilter() {

FilterRegistrationBean<CorsFilter> bean = new FilterRegistrationBean<>();

bean .set Filter(new CorsFilter());

bean .setOrder(Ordered .HIGHEST\_PRECEDENCE);

return bean;

}

3. 动态路由 ：



@RefreshScope

@Bean

public Zuul Filter dynamicRouteFilter() {

return new Zuul Filter() {

@Override

public Object run() {

// 从配置中心读取路由规则

}

};

}

核心优势 ：

 完全控制过滤器链

 避免服务发现和负载均衡开销  适合API聚合和边缘服务场景  可集成自定义路由逻辑

# **Zuul** 过滤器禁用配置指南

**1.** 基础禁用配置模板

zuul :

# 禁用特定过滤器

SendResponseFilter :

post :

disable : true # 禁用响应发送过滤器

PreDecorationFilter :

pre :

disable : false # 保持启用(示例)

**2.** 完整过滤器控制方案

zuul :

filter :

# 禁用内置过滤器

disable :

pre :

- ServletDetectionFilter # 禁用Servlet检测过滤器

- FormBodyWrapperFilter # 禁用表单包装器 post :

- SendResponseFilter # 禁用响应发送

- SendErrorFilter # 禁用错误发送

# 启用自定义过滤器 enable :

custom :

- AuthFilter # 自定义认证过滤器

- RateLimitFilter # 自定义限流过滤器

# 按路由禁用过滤器 routes :

api-service :

path : /api/\*\*

disabled-filters :

- FormBodyWrapperFilter # 该路由禁用表单过滤器

**3.** 内置过滤器清单



**4.** 生产环境推荐配置

zuul :

# 安全相关过滤器控制

SecureHeadersFilter :

pre :

disable : false # 强制启用安全头过滤器

# 性能相关过滤器控制

RibbonRoutingFilter :

route :

disable : ${DISABLE\_RIBBON:false} # 动态控制

# 按环境配置 profiles :

production :

disable-filters :

- DebugFilter

- Servlet30WrapperFilter

development :

disable-filters : []

**5.** 测试方案

# 1 . 验证过滤器状态

curl -s [http://zuul:8080/actuator/filters | jq](http://zuul:8080/actuator/filters|jq) '.disabled'

# 2 . 测试特定过滤器效果

curl -v -H "X-Disable-Filter: SendResponseFilter" [http://zuul:8080/api](http://zuul:8080/api#3.)

[# 3.](http://zuul:8080/api#3.) 动态禁用过滤器（需配合RefreshScope ）

curl -X POST -d 'zuul .SendResponseFilter .post .disable=true ' \

<http://zuul:8080/actuator/refresh>

**6.** 企业级管理方案

// 动态过滤器控制器

@RestController

@RequestMapping("/filters")

public class FilterAdminController {

@PostMapping("/{filterClass}/disable")

public String disableFilter(@PathVariable String filterClass) {

System .setProperty("zuul ."+filterClass+" .disable", "true");

return "Filter disabled";

}

}

**7.** 监控配置

management :

metrics :

tags :

filter : "${filter .class .simpleName}"

gauge :

zuul .filter .enabled :

enabled : true

endpoint :

filters :

enabled : true

show-disabled : true

**8.** 安全注意事项

security :

filter :

# 禁止禁用关键安全过滤器 protected-filters :

- SecureHeadersFilter

- CsrfHeaderFilter

# 过滤器操作审计 audit :

enabled : true

log-changes : true

**9.** 最佳实践建议

1. 分层控制 ：

zuul :

filter :

global :

disable :

- DebugFilter

route-specific :

/api/\*\* :

disable :

- FormBodyWrapperFilter

2. 自动化检查 ：

# 检查是否有危险配置

grep -r "zuul .\*disable=true" src/ --include="\* .yml"

3. 文档化策略 ：

## 禁用过滤器规范

- 必须记录所有禁用的过滤器

- 禁止在生产环境禁用安全过滤器

- 修改需通过CR审查

**10.** 故障排查命令

# 1 . 查看已禁用过滤器

curl -s [http://zuul:8080/actuator/filters | jq](http://zuul:8080/actuator/filters|jq) '.disabled'

# 2 . 检查过滤器执行链



curl -v [http://zuul:8080/api -H](http://zuul:8080/api-H) "X-Filter-Trace: true"

# 3 . 强制启用关键过滤器（调试用 ）

curl -X PUT <http://zuul:8080/filters/SecureHeadersFilter/enable>

关键安全规则 ：

1. 永远禁止禁用 SecureHeadersFilter 和 CsrfHeaderFilter

2. 生产环境禁止禁用 Send ErrorFilter

3. 所有过滤器变更必须经过性能测试

4. 维护完整的过滤器变更日志

5. 为每个禁用操作添加注释说明原因

# **Zuul Hystrix** 熔断降级配置指南

**1.** 基础降级配置模板

# application .yml

zuul :

routes :

customer-service :

path : /customers/\*\*

serviceId : customer-service

fallback :

enabled : true

path : /fallback/customers # 降级端点路径

hystrix :

command :

customer-service :

execution :

isolation :

thread :

timeoutInMilliseconds : 5000

circuitBreaker :

errorThresholdPercentage : 50

requestVolumeThreshold : 20

**2.** 完整降级方案

// 定制化降级提供器 @Component

public class CustomFallbackProvider implements FallbackProvider {

@Override

public String getRoute() {

return "customer-service"; // 指定服务ID }

@Override

public Client[HttpResponse fallbackResponse](HttpResponsefallbackResponse)(String route, Throwable cause) {

return new Client<HttpResponse>() {

@Override

public [HttpStatus getStatusCode](HttpStatusgetStatusCode)() {

return cause instanceof HystrixTimeoutException ?

[HttpStatus.GATEWAY\_TIMEOUT : HttpStatus.SERVICE\_UNAVAILABLE](HttpStatus.GATEWAY_TIMEOUT:HttpStatus.SERVICE_UNAVAILABLE);

}

@Override

public InputStream getBody() {

return new ByteArrayInputStream(

("{\"code\" :503,\"message\" :\"服务暂时不可用 : " +

cause .getMessage() + "\"}") .getBytes());

}

@Override

public [HttpHeaders getHeaders](HttpHeadersgetHeaders)() {

[HttpHeaders headers = new HttpHeaders](HttpHeadersheaders=newHttpHeaders)();

headers .setContentType(MediaType .APPLICATION\_JSON);

return headers;

}

};

}

}

**3.** 全局降级配置

// 全局降级处理器 @Component

public class Global FallbackProvider implements FallbackProvider {

@Override

public String getRoute() {

return "\*"; // 匹配所有路由 }

@Override

public Client[HttpResponse fallbackResponse](HttpResponsefallbackResponse)(String route, Throwable cause) {

return new Client<HttpResponse>() {

@Override

public [HttpStatus getStatusCode](HttpStatusgetStatusCode)() {

return determineStatus(cause);

}

// 其他必须实现的方法 . . .

};

}

}

**4.** 生产级配置建议

hystrix :

command :

default :

execution :

isolation :

thread :

timeoutInMilliseconds : 10000

circuitBreaker :

sleepWindowInMilliseconds : 10000

forceClosed : false

zuul :

fallback :

enable-stats : true # 启用降级统计 default-response :

code : 200

body : '{"status" :"DEGRADED"} '

routes :

critical-service :

fallbackUri : forward :/fallback/critical

payment-service :

fallbackUri : classpath :/static/fallback .json

**5.** 降级测试方案

# 1 . 触发熔断测试

for i in {1 . .50}; do

curl -X POST [http://zuul:8080/customers/timeout](http://zuul:8080/customers/123#3.)

[done](http://zuul:8080/customers/123#3.)

# 2 . 验证降级响应

curl -v [http://zuul:8080/customers/123](http://zuul:8080/customers/123#3.)

[# 3.](http://zuul:8080/customers/123#3.) 查看熔断状态

curl -s [http://zuul:8080/actuator/hystrix.stream | grep circuitBreaker](http://zuul:8080/actuator/hystrix.stream|grepcircuitBreaker)

**6.** 监控配置

management :

endpoints :

web :

exposure :

include : hystrix .stream

metrics :

distribution :

percentiles :

zuul .fallback .latency : 0 .95,0 .99

tags :

route : "${route .id}"

fallback : "${fallback .reason}"

**7.** 企业级架构方案

[客户端] → [Zuul网关] → [熔断检测] → [降级处理器]

↓

[监控系统] ← [配置中心]

**8.** 最佳实践建议

1. 分级降级 ：

public Client[HttpResponse fallbackResponse](HttpResponsefallbackResponse)() {

if (isCriticalRoute(route)) {

return critical Fallback();

} else {

return standard Fallback();

}

}

2. 动态降级 ：

@RefreshScope

@Bean

public FallbackProvider dynamicFallback() {

return new DynamicFallbackProvider(config);

}

3. 降级内容管理 ：

zuul :

fallback :

templates :

timeout : classpath :/templates/timeout .json

error : classpath :/templates/error .json

**9.** 安全防护措施

public class SecureFallbackProvider implements FallbackProvider {

@Override

public Client[HttpResponse fallbackResponse](HttpResponsefallbackResponse)() {

return new Client<HttpResponse>() {

@Override

public [HttpHeaders getHeaders](HttpHeadersgetHeaders)() {

[HttpHeaders headers = new HttpHeaders](HttpHeadersheaders=newHttpHeaders)();

headers .set("X-Content-Type-Options", "nosniff");

return headers;

}

};

}

}

**10.** 故障排查命令



# 1 . 查看熔断状态

watch -n 1 'curl -s [http://zuul:8080/actuator/health | jq .hystrix](http://zuul:8080/actuator/health|jq.hystrix) '

# 2 . 强制触发降级（测试用 ）

curl -H "X-Trigger-Fallback: true" [http://zuul:8080/api](http://zuul:8080/api#3.)

[# 3.](http://zuul:8080/api#3.) 验证降级内容

diff <(curl -s <http://zuul:8080/fallback/template>) template.json

关键规则 ：

1. 核心服务必须实现个性化降级

2. 降级响应不得包含敏感信息

3. 所有降级响应必须设置正确的Content-Type

4. 记录所有降级事件

5. 定期测试降级流程

# Zuul 超时配置指南

**1.** 基础超时配置模板

# 服务发现模式下的超时配置 ribbon :

ReadTimeout : 10000 # 读取超时(ms)

ConnectTimeout : 2000 # 连接超时(ms)

# 直连URL模式下的超时配置 zuul :

host :

socket-timeout-millis : 15000 # 响应超时 connect-timeout-millis : 3000 # 连接超时

**2.** 生产级超时方案

# 全局默认超时 hystrix :

command :

default :

execution :

isolation :

thread :

timeoutInMilliseconds : 30000 # Hystrix熔断超时

# 服务级超时配置 ribbon :

eureka :

enabled : true

ReadTimeout : ${SERVICE\_TIMEOUT:10000}

ConnectTimeout : 2000

MaxAutoRetries : 1

MaxAutoRetriesNextServer : 2

# 关键服务独立配置 customer-service :

ribbon :

ReadTimeout : 30000 # 客户服务特殊超时

ConnectTimeout : 5000

**3.** 超时配置矩阵



**4.** 动态超时配置

@Configuration

public class DynamicTimeoutConfig {

@Bean

@RefreshScope

public IClientConfig ribbonClientConfig() {

return new DefaultClientConfig Impl() {

@Override

public int getPropertyAsInteger(String key, int defaultValue) {

if ("ReadTimeout" .equals(key)) {

return getDynamicTimeout(); // 从配置中心读取

}

return super .getPropertyAsInteger(key, defaultValue);

}

};

}

}

**5.** 测试方案

# 1 . 测试连接超时

timeout 1s curl -v [http://zuul:8080/api/slow-connect](http://zuul:8080/api/data#3.)

[# 2.](http://zuul:8080/api/data#3.) 测试读取超时

curl -H "X-Simulate-Delay: 15s" [http://zuul:8080/api/data](http://zuul:8080/api/data#3.)

[# 3.](http://zuul:8080/api/data#3.) 验证熔断触发

watch -n 1 'curl -s [http://zuul:8080/actuator/hystrix.stream | grep timeout](http://zuul:8080/actuator/hystrix.stream|greptimeout) '

# 4 . 查看当前超时设置

curl -s [http://zuul:8080/actuator/env/ribbon.ReadTimeout | jq](http://zuul:8080/actuator/env/ribbon.ReadTimeout|jq)

**6.** 监控配置

management :

metrics :

distribution :

percentiles :

zuul .timeout .requests : 0 .95,0 .99

tags :

timeout .type : "${timeout .source}"

endpoint :

timeout :

enabled : true

cache :

time-to-live : 10s

**7.** 企业级最佳实践

1. 分层超时策略 ：

timeout :

tiers :

critical :

read : 30000

connect : 5000

standard :

read : 10000

connect : 2000

2. 服务画像 ：

public class ServiceProfile {

@Scheduled(fixedRate = 60000)

public void adjustTimeouts() {

// 根据服务性能动态调整超时

}

}

3. 异常处理 ：

@ControllerAdvice

public class TimeoutHandler {

@ExceptionHandler(HystrixTimeoutException .class)

public ResponseEntity<String> handleTimeout() {

return ResponseEntity.status(504)

.body("{\"error\" :\"Gateway Timeout\"}");

}

}

**8.** 故障排查命令

# 1 . 查看实际生效超时

curl -s [http://zuul:8080/actuator/configprops | jq](http://zuul:8080/actuator/configprops|jq) '.ribbon'

# 2 . 触发超时告警测试

curl -H "X-Trigger-Timeout: 15000" [http://zuul:8080/test](http://zuul:8080/test#3.)

[# 3.](http://zuul:8080/test#3.) 监控线程池状态

watch -n 1 'curl -s [http://zuul:8080/actuator/metrics | grep](http://zuul:8080/actuator/metrics|grep) "hystrix.threadpool"'

**9.** 安全配置

security :

timeout :

max-limit : 60000 # 最大允许超时

min-limit : 1000 # 最小允许超时

audit :

enabled : true # 记录超时配置变更

**10.** 多环境配置

spring :

profiles : production



ribbon :

ReadTimeout : 30000

---

spring :

profiles : staging

ribbon :

ReadTimeout : 60000 # 测试环境延长超时

---

spring :

profiles : development

ribbon :

ReadTimeout : 0 # 开发环境禁用超时

关键规则 ：

1. Hystrix超时 > Ribbon超时 + 重试时间

2. 关键服务独立配置超时

3. 生产环境超时不低于3秒

4. 定期审计超时配置

5. 实现超时分级告警

# **Zuul Location Header** 重写配置指南

**1.** 基础配置模板

@Configuration

@EnableZuulProxy

public class ZuulConfig {

/\*\*

\* 注册Location重写过滤器

\* 作用 ：将后端服务的重定向URL改写为Zuul网关URL \*/

@Bean

public LocationRewriteFilter locationRewriteFilter() {

return new LocationRewriteFilter();

}

}

**2.** 生产级安全配置

zuul :

location :

rewrite :

enabled : true

include-routes :

- user-service # 仅对指定路由生效 exclude-paths :

- /external/\*\* # 排除外部URL重定向 strict-mode : true # 严格校验URL格式

**3.** 高级控制方案

@Bean

public LocationRewriteFilter customLocationFilter() {

return new LocationRewriteFilter() {

@Override

public boolean shouldFilter([HttpServletResponse response](HttpServletResponseresponse)) {

// 仅处理302重定向且路径匹配的请求

return response .getStatus() == 302 &&

RequestContext .getCurrentContext()

.getRequest() .getRequestURI() .startsWith("/api/");

}

@Override

public String rewriteLocation(String original Location) { // 自定义重写逻辑

return original Location .replace("internal", "gateway");

}

};

}

**4.** 配置详解



**5.** 测试方案

# 1 . 测试内部重定向

curl -v [http://zuul:8080/api/login #](http://zuul:8080/api/login#) 应返回Zuul格式的Location头

# 2 . 测试外部URL排除

curl -v [http://zuul:8080/api/redirect?target=http://external.com](http://zuul:8080/api/redirect?target=http://external.com#3.)

[# 3.](http://zuul:8080/api/redirect?target=http://external.com#3.) 验证重写效果

curl -s [http://zuul:8080/api/test-redirect | grep -i location](http://zuul:8080/api/test-redirect|grep-ilocation)

**6.** 监控与安全

management :

metrics :

tags :

location .rewrite : "${rewrite .status}"

endpoint :

redirects :

enabled : true

sensitive : true

security :

location :

allowed-domains :

- yourdomain .com

block-external : true

**7.** 企业级架构建议

[浏览器] → [Zuul网关] → [应用服务]

↑ ↓

└─────[重定向URL改写]←─┘

**8.** 最佳实践

1. 白名单控制 ：

@Bean

public FilterRegistrationBean<UrlValidatorFilter> urlValidationFilter() {

FilterRegistrationBean<UrlValidatorFilter> bean = new FilterRegistrationBean<>();

bean .set Filter(new UrlValidatorFilter());

bean .addUrlPatterns("/\*");

return bean;

}

2. 动态配置 ：

@RefreshScope

@Bean

public LocationRewriteFilter dynamicFilter(

@Value("${zuul .location .rewrite .rules}") String rules) {

return new DynamicLocationFilter(rules);

}

3. 审计日志 ：

public class LocationAuditFilter extends Zuul Filter {

@Override

public Object run() {

auditLog.log(RequestContext .getCurrentContext());

return null;

}

}

**9.** 故障排查

# 1 . 查看重定向历史

curl -s [http://zuul:8080/actuator/redirects | jq](http://zuul:8080/api/auth#3.)

[# 2.](http://zuul:8080/api/auth#3.) 调试过滤器链

curl -H "X-Debug-Filters: true" [http://zuul:8080/api/auth](http://zuul:8080/api/auth#3.)

[# 3.](http://zuul:8080/api/auth#3.) 验证安全规则

curl -v "<http://zuul:8080/redirect?url=http://malicious.com>"

**10.** 紧急控制方案



# 临时禁用重写功能 zuul :

location :

rewrite :

enabled : false

emergency-mode : true # 记录但不改写

关键安全警告 ：

1. 必须配置 exclude-paths 排除外部域名

2. 重写前验证URL的合法性

3. 监控所有302/303响应

4. 禁止处理包含 ../ 的路径

5. 定期审计重定向日志

# **Zuul** 跨域请求**(CORS)**配置指南

**1.** 基础**CORS**配置模板

@Configuration

public class ZuulCorsConfig {

/\*\*

\* 全局CORS配置

\*/

@Bean

public WebMvcConfigurer corsConfigurer() {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowedMethods("GET", "POST", "PUT", "DELETE")

.allowedHeaders("\*")

.allowCredentials(false)

.maxAge(3600);

}

};

}

}

**2.** 生产级**CORS**方案

# application .yml zuul :

cors :

enabled : true

global-config : # 全局默认配置

allowed-origins: "<https://yourdomain.com>"

allowed-methods : "GET,POST,PUT,DELETE"

allowed-headers : "Content-Type,Authorization"

exposed-headers : "X-Custom-Header"

allow-credentials : true

max-age : 1800

route-configs : # 路由级覆盖配置

user-service :

allowed-origins: "<https://user.yourdomain.com>"

allowed-methods : "GET,POST"

admin-service :

allowed-origins: "<https://admin.yourdomain.com>"

allowed-headers : "X-Auth-Token"

**3.** 多环境**CORS**配置

@Profile("production")

@Bean

public WebMvcConfigurer productionCorsConfig() {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/api/\*\*")

.allowedOrigins("<https://prod.yourdomain.com>")

.allowCredentials(true);

}

};

}

@Profile("development")

@Bean

public WebMvcConfigurer devCorsConfig() {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins("\*")

.allowedMethods("\*");

}

};

}

**4.** 动态**CORS**控制

@RefreshScope

@Bean

public WebMvcConfigurer dynamicCorsConfig(

@Value("${cors .allowed .origins}") String[] origins) {

return new WebMvcConfigurer() {

@Override

public void addCorsMappings(CorsRegistry registry) {

registry.addMapping("/\*\*")

.allowedOrigins(origins)

.allowedMethods("GET", "POST");

}

};

}

**5.** 安全增强配置

security :

cors :

origin-patterns : # 正则表达式匹配

- "^<https://>[a-z0-9-]+\\.yourdomain\\.com$"

strict-transport : true

preflight-cache :

size : 1000

ttl : 1h

**6.** 测试方案

# 1 . 测试预检请求

curl -X OPTIONS -H "Origin: <https://test.yourdomain.com>" \

-H "Access-Control-Request-Method : POST" \

[http://zuul:8080/api/users -v](http://zuul:8080/api/users-v#2.)

[# 2.](http://zuul:8080/api/users-v#2.) 验证CORS头

curl -s -H "Origin: <https://invalid.com>" \

[http://zuul:8080/api | grep -i](http://zuul:8080/api|grep-i) "Access-Control"

# 3 . 测试带凭证请求

curl -H "Origin: <https://user.yourdomain.com>" \

-H "Cookie : sessionId=123" \

[http://zuul:8080/auth -v](http://zuul:8080/auth-v)

**7.** 监控配置

management :

metrics :

tags :

cors .origin : "${request .headers .origin}"

distribution :

percentiles :

zuul .cors .requests : 0 .95,0 .99

endpoint :

cors :

enabled : true

show-denied : true

**8.** 企业级架构建议

[浏览器] → [Zuul网关CORS处理] → [微服务]

↓

[CORS策略中心] ← [配置中心]

**9.** 最佳实践

1. 分层控制 ：

registry.addMapping("/public/\*\*")

.allowedOrigins("\*");

registry.addMapping("/internal/\*\*")

.allowedOrigins("<https://internal.com>");

2. 安全审计 ：

public class CorsAuditFilter extends Zuul Filter {

@Override

public Object run() {

auditCorsRequest(RequestContext .getCurrentContext());

return null;

}

}

3. 动态更新 ：

# 动态刷新CORS配置

curl -X POST [http://zuul:8080/actuator/refresh -d](http://zuul:8080/actuator/refresh-d) 'cors.allowed.origins=[https://newdoma](http://zuul:8080/actuator/cors-mappings|jq#2.) [in.com](http://zuul:8080/actuator/cors-mappings|jq#2.) '

**10.** 故障排查



# 1 . 查看当前CORS配置

curl -s [http://zuul:8080/actuator/cors-mappings | jq](http://zuul:8080/actuator/cors-mappings|jq#2.)

[# 2.](http://zuul:8080/actuator/cors-mappings|jq#2.) 模拟非法源请求

curl -H "Origin: <http://attacker.com>" -v [http://zuul:8080/api](http://zuul:8080/actuator/cors-mappings|jq#2.)

[# 3.](http://zuul:8080/actuator/cors-mappings|jq#2.) 检查CORS过滤器链

curl -H "X-Debug-CORS: true" <http://zuul:8080/api>

关键安全规则 ：

1. 生产环境禁止使用 allowedOrigins: "\*"

2. 敏感接口必须设置 allowCredentials: false

3. 严格限制 allowed Methods 范围

4. 监控异常的 Origin 头

5. 定期审计CORS策略

# **Zuul** 网关监控指标配置指南

**1.** 基础监控配置

management :

endpoints :

web :

exposure :

include : health,metrics,zuul

metrics :

tags :

application : ${spring.application .name}

distribution :

percentiles :

zuul .request .time : 0 .95,0 .99

**2.** 关键指标说明



**3.** 生产级监控方案

management :

metrics :

export :

prometheus :

enabled : true

step : 1m

enable :

zuul : true

hystrix : true

[http: true](http:trueweb:)

[web:](http:trueweb:)

server :

auto-time-requests : true

tags :

route : "${route Id}"

status : "${status}"

exception : "${exception}"

**4.** 自定义指标扩展

@Bean

public ZuulMetricsFilter zuulMetricsFilter(MeterRegistry registry) {

return new ZuulMetricsFilter(registry) {

@Override

public void recordError(Throwable ex) {

// 自定义异常指标

registry.counter("zuul .custom .errors",

"type", ex .getClass() .getSimpleName()) .increment();

}

};

}

**5.** 告警规则示例

alerts :

zuul :

error-rate :

threshold : 5% # 错误率阈值

window : 5m

latency-p99 :

threshold : 2s

circuit-breaker :

open :

severity : critical

**6.** 监控端点测试

# 1 . 查看所有指标

curl -s [http://zuul:8080/actuator/metrics | jq](http://zuul:8080/actuator/hystrix.stream#4.)

[# 2.](http://zuul:8080/actuator/hystrix.stream#4.) 查询特定异常指标

curl -s "<http://zuul:8080/actuator/metrics/ZUUL::EXCEPTION>" | jq

# 3 . 实时监控Hystrix流

curl [http://zuul:8080/actuator/hystrix.stream](http://zuul:8080/actuator/hystrix.stream#4.)

[# 4.](http://zuul:8080/actuator/hystrix.stream#4.) 性能压力测试

ab -n 1000 -c 100 <http://zuul:8080/api/test>

**7. Grafana**仪表板配置

# 请求量统计

SELECT sum(rate(zuul\_requests\_total[1m]))

FROM metrics

GROUP BY route Id

# 错误率计算

SELECT sum(zuul\_exception\_errors) / sum(zuul\_requests\_total)

FROM metrics

WHERE status != '200 '

**8.** 企业级监控架构

[Zuul集群] → [Micrometer指标] → [Prometheus] → [Grafana]

↓

[异常日志] → [ELK]

**9.** 最佳实践建议

1. 分层监控 ：

threshold : 1%

monitoring :

levels :

critical :

routes :

- payment

- auth

normal :

threshold : 5%

2. 动态采样 ：

@Bean

public MeterFilter samplingFilter() {

return MeterFilter .sample(

Bucket Filter .sample(100,

req -> req.getUri() .contains("/api/"));

}

3. 指标聚合 ：

@Bean

public TimedAspect timedAspect(MeterRegistry registry) {

return new TimedAspect(registry);

}

**10.** 故障排查工具



# 1 . 实时指标监控

watch -n 1 'curl -s [http://zuul:8080/actuator/metrics/zuul.requests | jq](http://zuul:8080/actuator/metrics/zuul.requests|jq) '

# 2 . 异常追踪

curl -s "<http://zuul:8080/actuator/metrics/ZUUL::EXCEPTION:500>" | jq

# 3 . 线程池状态

curl -s [http://zuul:8080/actuator/metrics/hystrix.threadpool | jq](http://zuul:8080/actuator/metrics/hystrix.threadpool|jq)

关键监控项 ：

1. ZUUL::EXCEPTION:500 - 服务端错误

2. zuul.route.requests - 路由流量分布

3. hystrix.circuit.breaker.open - 熔断器状态

4. [http.server.requests - HTTP](http.server.requests-HTTP)请求指标

5. zuul.filter.latency - 过滤器耗时

# **Zuul** 深度开发指南

**1.** 核心架构配置

# application .yml zuul :

servlet-path : /gateway # 修改默认Servlet路径 ribbon :

eager-load :

enabled : true # 启用Ribbon预加载 debug :

request : true # 开启调试模式

**2.** 过滤器开发模板

// 预处理过滤器示例

public class AuthPreFilter extends Zuul Filter {

@Override

public String filterType() {

return PRE\_TYPE; // 预处理阶段

}

@Override

public int filterOrder() {

return 5; // 执行顺序 }

@Override

public boolean shouldFilter() {

return !RequestContext .getCurrentContext()

.getZuulRequestHeaders()

.containsKey("X-Auth-Skip");

}

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

if ( !validateToken(ctx .getRequest())) {

ctx .setSendZuulResponse(false);

ctx .setResponseStatusCode(401);

}

return null;

}

}

**3.** 路由过滤器对比

| **​**过滤器类型**​** | **​**典型实现**​** | **​**执行阶段**​** | **​**开发要点**​** |
| --- | --- | --- | --- |
| Pre | PreDecorationFilter | 路由决策前 | 设置路由关键参数 |
| Route | RibbonRoutingFilter | 请求转发时 | 处理HTTP客户端交互 |
| Post | SendResponseFilter | 响应返回前 | 修改响应头和内容 |
| Error | SendErrorFilter | 异常发生时 | 统一错误格式处理 |

**4.** 生产级路由配置

@Bean

public RouteLocator customRouteLocator(RouteLocatorBuilder builder) {

return builder .routes()

.route("dynamic-route", r -> r .path("/dynamic/\*\*")

.filters(f -> f .addRequestHeader("X-Route-Type", "dynamic"))

.uri("lb ://dynamic-service"))

.route("legacy-route", r -> r .path("/legacy/\*\*")

.filters(f -> f .rewritePath("/legacy/(?<segment> .\*)", "/${segment}"))

.uri("<http://legacy.example.com>"))

.build(); }

**5.** 异常处理方案

@Bean

public ErrorFilter errorFilter() {

return new ErrorFilter() {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

ctx .set("error .status\_code", ctx .getResponseStatusCode());

ctx .set("error .exception", ctx .getThrowable());

return null;

}

};

}

**6.** 性能优化配置

# 连接池配置

<httpclient:>

max-total-connections : 500

max-per-route-connections : 50

time-to-live : 60000

# 线程池配置

server :

tomcat :

max-threads : 200

min-spare-threads : 20

**7.** 监控集成方案

@Bean

public ZuulMetricsFilter metricsFilter(MeterRegistry registry) {

return new ZuulMetricsFilter(registry) {

@Override

public void recordRouteMetrics(RequestContext context) {

Timer .builder("zuul .route .timer")

.tags("route", context .get("proxy") .toString())

.register(registry)

.record( . . .);

}

};

}

**8.** 安全增强措施

public class SecurityPost Filter extends Zuul Filter {

@Override

public Object run() {

RequestContext ctx = RequestContext .getCurrentContext();

ctx .getResponse() .setHeader("X-Content-Type-Options", "nosniff");

ctx .getResponse() .setHeader("Content-Security-Policy", "default-src 'self '");

return null;

}

}

**9.** 动态配置示例

@RefreshScope

@Bean

public Zuul Filter dynamicFilter(

@Value("${filter .config}") FilterConfig config) {

return new Zuul Filter() {

@Override

public Object run() {

// 使用动态配置

}

};

}

**10.** 调试与测试工具

// 调试过滤器

@Bean

public DebugFilter debugFilter() {

return new DebugFilter() {

@Override

public boolean shouldFilter() {

return Boolean .parseBoolean(

RequestContext .getCurrentContext()

.getRequest() .getHeader("X-Debug"));

}

};

}

企业级开发规范 ：

1. 所有自定义过滤器必须实现 toString() 方法

2. 路由配置变更需通过配置中心管理

3. 生产环境禁用 debug.request 功能

4. 关键过滤器需有单元测试覆盖

5. 维护过滤器执行顺序文档

**11.** 请求上下文最佳实践

// 安全访问RequestContext

public class ContextHelper {

public static void safePut(String key, Object value) {

RequestContext ctx = RequestContext .getCurrentContext();

if ( !ctx .containsKey(key)) {

ctx .put(key, value);

}

}

}

**12.** 版本迁移策略

zuul :

migration :

v1 :

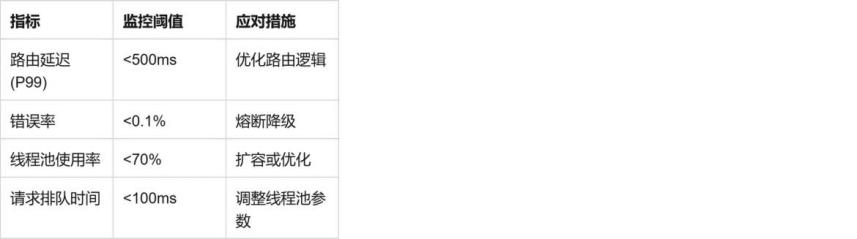
path : /v1/\*\*

strategy : shadow # 影子流量测试 v2 :

path : /v2/\*\*

strategy : canary # 灰度发布

**13.** 性能关键指标



**14.** 扩展阅读

. [Zuul 过滤器生命周期](https://github.com/Netflix/zuul/wiki/Filter-Lifecycle)

. [Spring Cloud Circuit Breaker](https://spring.io/projects/spring-cloud-circuitbreaker)

. [Micrometer监控集成](https://micrometer.io/docs/registry/prometheus)